Министерство просвещения РФ
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Уральский государственный педагогический университет»

Институт математики, физики, информатики и технологий

Кафедра информатики, информационных технологий
и методики обучения информатике

МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ ОРГАНИЗАЦИИ СОВМЕСТНЫХ ВЕЛОПОХОДОВ

Выпускная квалификационная работа
бакалавра по направлению подготовки
09.03.03 – Прикладная информатика в сервисе

Исполнитель: студент группы ПИ 1601z
Института математики, физики,
информатики и технологий
Патраков А.В.

Работа допущена к защите
«____»_____ 2021 г.
Зав. кафедрой _____

Руководитель: кандидат педагогических наук,
доцент кафедры ИИТиМОИ
Кудрявцев А.В.

Екатеринбург – 2021

Реферат

Патраков А.В. МОБИЛЬНОЕ ПРИЛОЖЕНИЕ ДЛЯ ОРГАНИЗАЦИИ СОВМЕСТНЫХ ВЕЛОПОХОДОВ, выпускная квалификационная работа: 58 стр., рис. 43, табл. 1, библ. 31 назв.

Ключевые слова: БАЗА ДАННЫХ, ПРОГРМАММНОЕ ОБЕСПЕЧЕНИЕ, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, ИНТЕРФЕЙС, RESTAPI.

Продукт разработки – мобильное приложение для организации совместных велопоходов.

Цель работы – разработка мобильного приложения, обеспечивающего удобную организацию совместных велопоходов.

В работе описаны результаты проектирования и разработки мобильного приложения, при использовании современных средств и методов проектирования и разработки.

В ходе разработки использовались такие инструменты как: язык программирования PHP, язык программирования Java Script, IDE PhpStorm, PHP Framework Yii2, React Framework React Native, IDE Visual Studio Code, система контроля версий и хранилище исходников проекта Git.

Исходный код приложения располагается в приватных репозиториях на GitHub https://github.com/Patrick20191020/letsRide-front-end
и https://github.com/Patrick20191020/letsRide-back-end. Проект базы данных - https://app.dbdesigner.net/designer/schema/391000.

Оглавление

Введение

Мобильные устройства, неотъемлемая часть жизни современного человека. Мобильные устройства широко внедряются в сферу сервиса, в связи, с чем появляется потребность в разработке мобильных приложений, оптимизирующих деятельность человека.

На текущий день, работу разрабатываемого мобильного приложения заменяют группы в социальных сетях, но их большое количество, из-за чего проявляются проблемы:

1. Трудно размещать информацию, из-за множества групп.
2. Отсутствует прозрачность мероприятия, так как участники набираются из разных групп.
3. Вероятность отсутствия актуальности мероприятия.
4. Долговременное донесение информации до объекта.
5. Сложная организованность мероприятия.

Актуальность данного мобильного приложения подтверждается решением описанных проблем. Мобильное приложение будет выступать единой точкой входа и выхода информации. Будет удобнее организовывать процесс, удобнее вносить корректировки, в свою очередь участникам будет легче находить необходимую информацию и также отслеживать внесенные изменения в мероприятие.

Цель разработки – разработать мобильное приложение для организации совместных велопоходов.

Задачи:

1. Выполнить анализ состояния проблемы и подходов к ее решению.
2. Обосновать выбор технологий реализации и необходимых программных платформ.
3. В соответствии с техническим заданием провести разработку мобильного приложения.

Глава 1. Теоретико-аналитическая часть

1.1 Обоснование постановки задачи

Мобильное приложение — это программное обеспечение, специально разработанное под конкретную мобильную платформу (iOS, Android, Windows Phone и т. д.). Предназначено для использования на смартфонах, фаблетах, планшетах, умных часах и других мобильных устройствах [1, 27].

Программное обеспечение (далее «ПО») - это взаимодействие каждой из частей системы логической цепочки нулей и единиц, работающих по определённому алгоритму обработки и работы с информацией, которые так же могут являться программами [2]. То есть, если говорить простыми словами, то это программа, разработанная на каком-то языке программирования, в которой зашит алгоритм ее работы.

Мобильная платформа – это операционная система для телефонов по аналогии с компьютерами [3].

На текущий день, активную популярность набирает активный спорт катание на велосипедах. Для организации совместного катания, а также участия в таком мероприятии, используется: сарафанное радио, объявления в интернете, в частности в группах в социальной сети. Но таких групп просто огромное количество, приходится смотреть все группы, все предложения и на это тратятся колоссальные усилия, что даже желание покататься просто пропадает. Да и сарафанное радио не несет полной достоверности информации.

Таким образом, возникают проблемы:

1. Достоверность информации – не актуализируется информация, после внесение корректировок.
2. Доступность информации – отсутствие социальных сетей.
3. Прозрачность информации – нет четкого понимания, сколько и кто именно является участником или организатором.

4. Отсутствие онлайн общения посредством телефона с организатором мероприятия.

5. Для организации мероприятия необходимо потратить немало усилий.

После выявления проблемных мест, появляется два варианта решения – это создание сайта или создание мобильного приложения.

Рассмотрим преимущества мобильного приложения по отношению к сайту:

1. Производительность – приложение достаточно единожды установить на телефон для работы с ним. После работа с приложением будет в два действия: открыть телефон, открыть приложение. Для использования сайта необходимо выполнять три действия: открыть телефон, открыть браузер, открыть сайт. Дополнительный фактор выбора мобильного приложения, браузеры на сегодняшний день очень ресурсоемкие.

2. Доступность донесения информации – при помощи мобильного приложения достаточно легко размещать и читать информацию. Дополнительно возможно реализовать push-уведомления, которые сократят время донесения информации до объекта.

Исходя из вышенаписанного, выбор был сделан в сторону мобильного приложения. После реализации мобильного приложения появится возможность в одном месте находить подходящие маршруты, поездки, можно переписываться с организатором, что облегчит организованность и согласованность маршрута. Звонить, увидеть всех участников, увидеть фото участников, стать самому организатором поездки и самое главное, это всегда под рукой в телефоне.

1.2 Выбор средств и метода разработки

После принятия окончательного решения о начале разработки мобильного приложения появляется вопрос, касаемо его архитектуры, какую именно архитектуру реализации программного обеспечения выбрать, а также выбора языков программирования, необходимых Фреймворков, программных обеспечений и т.д.

Архитектура – это совокупность работы компонентов в программном обеспечении в едином целом [5, 28]. В разработке мобильного приложения была выбрана микросервисная архитектура.

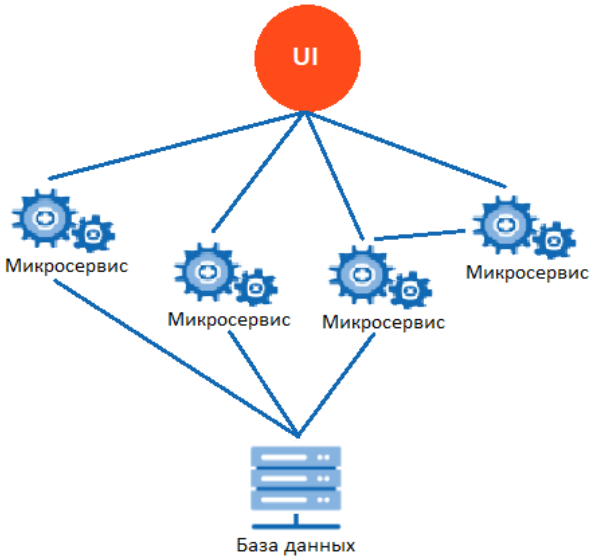

Рисунок 1- Микросервисная архитектура

Микросервисная архитектура реализуется на основное небольших, модульных сервисов, каждый из которых выполняет свою уникальную задачу. Данные модули взаимодействуют (общаются) при помощи RESTAPI, с использованием формата JSON [6].

В свою очередь REST API (Representational State Transfer) – это общие принципы взаимодействия приложения с сервером [7, 29]. JSON – это формат для хранения и обмена информацией, в основном используется для передачи данных между сервером и пользователем [8].

Основные плюсы выбора микросервисной архитектуры ПО:

1. Упрощает внесения доработок/исправлений, то есть легче масштабировать приложение, так как сервисы не зависимы.
2. Удобен при разработке несколькими разработчиками, опять же из-за независимости сервисов.
3. При необходимости возможно одним действием отключить сервис, что позволяет снизить нагрузку на сервер.
4. Безопасность. Можно смело гарантировать, что передаваться будет только та информация, которая запрашивается, так как общение происходит по API.

После того, как была выбрана архитектура ПО, уже понятно, что будет наше мобильное приложение, будет делиться на 3 части: интерфейсная часть (front-end), API (back-end), база данных.

В первую очередь, было принято решение о выборе СУБД для дальнейшей работы с базой данных.

На текущий момент популярными СУБД являются:

1. Oracle.
2. MySQL.
3. Microsoft SQL Server.
4. Mongo DB.

Для того чтобы выбрать конкретную СУБДнам необходимы критерии, на основании чего и будет сделан соответствующий выбор:

1. База данных должна быть реляционной, где данные организованы в виде разных таблиц, а таблица состоит из столбцов (полей) и строк (записей).
2. СУБД предоставляется на бесплатной основе.
3. От компании, сопровождающих данные СУБД, должна быть доступна корректная и актуальная документации.

Для удобства и наглядности подходящих вариантов, исходя из вышеописанных критериев, представим сведения о СУБД в виде таблицы.

Таблица 1. Сравнение СУБД

Наименование СУБД	1 критерий	2 критерий	3 критерий
Oracle	+	-	+
MySQL	+	+	+
Microsoft SQL Server	+	-	+
Mongo DB	-	+	+

Исходя из приведенной таблицы, делаем выбор в пользу MySQL, так как эта СУБД соответствует всем заданным критериям. MySQL - это система управления базами данных, распространяемая как свободное программное обеспечение [9, 30, 31]. Дополнительно можно указать, что данная СУБД показала надежность в работе, возможность обеспечения безопасности данных, а также предоставляет удобный интерфейс для пользователя.

После выбора СУБД следующим шагом является реализация интерфейса программного обеспечения, а именно необходимо определится с выбором языка программирования или Фреймворка и среды программирования интерфейса. Для того, чтобы не писать интерфейс дважды под разные платформы мобильных устройств, будем рассматривать исключительно кроссплатформенные инструменты программирования.

Кроссплатформенность – это свойство страницы, позволяющее работать сразу на нескольких видах операционных систем или платформ [10].

Популярные инструменты:

1. Apache Cordova– мобильная среда разработки приложений, имеет второе народное название PhoneGap. Позволяет программистам создавать

приложения для мобильных устройств с помощью CSS3, HTML5 и JavaScript. Это обеспечивается за счет преобразования из CSS, HTML и JavaScript в код, который любая платформа воспринимает как элемент web. Выпущен в 2012 году. Основным минусом является ресурсоёмкость, вследствие чего приложение подтормаживает на слабых мобильных устройствах [11].

2. Flutter–среда разработки с открытым исходным кодом для разработки мобильных приложений выпущенный от компании Google. Данная среда была выпущена относительно недавно в 2018, ранее имела название Sky, и специализирована только под Android. К сожалению, многие минусы, проблемы были унаследованы от предыдущих версий [12]. На официальном сайте Google, для данного Фреймворка указано три основных плюса:

a. Быстрая разработка (присуще только для опытного разработчика);
b. Выразительный и гибкий интерфейс (Фреймворк сам умеет настраивать шрифты, расположение и физику скроллов и т.д.);
c. Производительность (собственный графический движок).

3. React Native – Фреймворк для разработки мобильных приложений, реализованный на основе React (это инструмент для создания пользовательских веб-страниц). Выпущен был в 2015 году. React Native популярный так как компактен и отличается высокой производительностью. Для написания кода используется JavaScript. Особую популярность придало использование крупными компаниями, такими как: Facebook, Instagram, Skype, Tesla и т.д., чтобы подтверждает его перспективу на будущие года [13].

Анализируя вышенаписанное, делаем вывод, что Flutter не подходит, так как еще находится на стадии разработки и усовершенствования. Также можно исключить Apache Cordova в связи с низкой производительностью, соответственно выбор падает на React Native. В дополнение к React Native можно сказать, что у него отсутствуют привычная html разметка, присутствуют свои, встроенные компоненты. Также присутствует всем привычный

формальный язык описания внешнего вида компонентов (CSS). В свою очередь CSS даст нам возможность настраивать компоненты полностью по нашему желанию.

Следующим и также немаловажным шагом является выбор языка программирования для back-end. Если в поисковой системе найти «рейтинг языков программирования back-end», лидером будет PHP и на втором месте .Net, причем отрыв будет колоссальным, так как на PHP реализовано более 79% процентов всех веб-сайтов [14]. Поэтому выбор очевиден в сторону PHP.

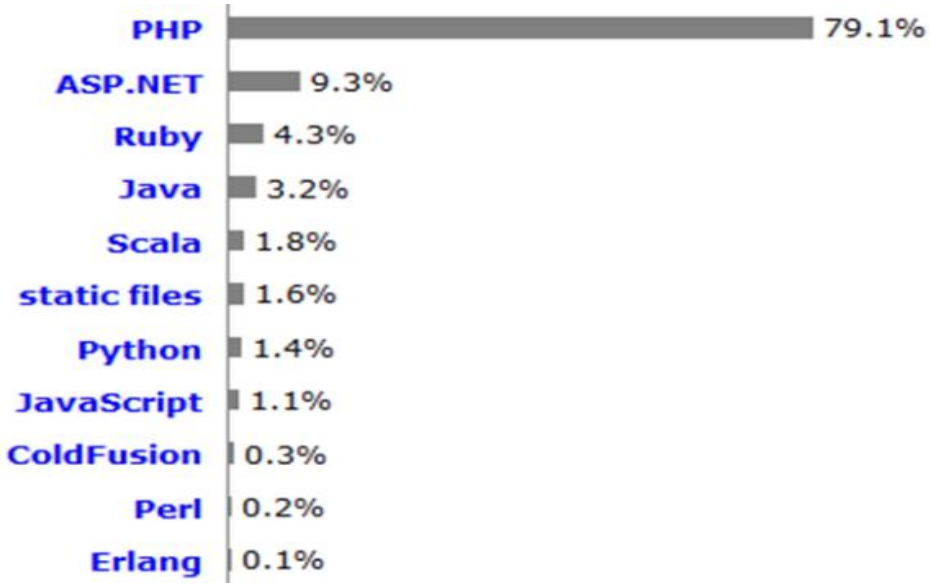

Рисунок 2 - Рейтинг языков программирования back-end

PHP - это распространённый язык программирования общего назначения с открытым исходным кодом. PHP специально сконструирован для веб-разработок и его код может внедряться непосредственно в HTML. Данный язык программирования был выпущен в 1995 году [15, 26].

Основные плюсы PHP:

1. Легко установить.
2. Полная документация.
3. Допускает собирать программное обеспечение с ошибками и будет выполняться, пока не достигает участка кода.
4. Для работы подходит любая платформа.

5. Бесплатный.

После выбора языков программирования необходимо определится со средой разработки. На текущий момент более 50 ПО для написания кода, причем часть из них специализированное, то есть предназначено для определенного языка, а часть являются универсальными. Рассмотрим наиболее подходящие для React Native и PHP.

Популярные редакторы для React-Native:

1. Nuclide – специализированная среда разработка с открытым кодом, представлен от компании Facebook. Бесплатный. Довольно простой и удобный интерфейс, представляет первоклассную среду разработки для проекта React Native, имеет встроенную поддержку данного Фреймворка. Более не сопровождается, находится в финальной версии. Отсутствует полноценная поддержка ОС Windows [16].

2. Sublime Text – проприетарный текстовый редактор. Документально является специализированный для языка программирования Python, но по факту поддерживаем программирование на любом. Позволяет программировать бесплатно, однако постоянно просить приобрести лицензию. Более не сопровождается [17].

3. Visual Studio Code – универсальный редактор исходного кода, представлен от компании Microsoft. На текущий день сопровождается. В редакторе присутствует функционал по установке компонентов, который позволяют разрабатывать на любом языке программировании. Бесплатный. Легкий и очень приятный, а также интуитивный интерфейс [18].

Очевидно, был выбран VS Code, который после установки дополнительных компонентов для Фреймворка React Native для оформления синтаксиста кода, позволяет легко работать в данном программном обеспечении.

Популярные редакторы для языка программирования PHP:

1. NetBeans – свободная интегрированная среда разработки приложений. Поддерживает множество языков программирования и большое количество Фреймворков. Данное программное обеспечение поддерживает русский язык. Первый выпуск 1997 год, на текущий момент сопровождение ведется разработчиками на добровольной основе, но спонсирование идет от компании Oracle [19].

2. PHPStorm – кросс платформенная среда разработки. Программное обеспечение является коммерческое, но для преподавателей и студентов ВУЗов предоставляется бесплатно, также присутствует 30-днейный бесплатный период эксплуатации, выпущено в 2009 году. Данное программное обеспечение реализовано на основе другой среды разработки IntelliJ IDEA и специализировано, исходя из названия по язык программирования PHP [20]. Дополнительным и главным плюсом является Xdebug, представляющий собой расширение для PHP, предназначенный для отладки кода [21].

Данные программные обеспечения являются равносильными и равноценными, каких-то жестких критериев перевеса на какое-то одно приложение нет, можно сказать, каждый выбирает то, что ему больше нравится по дизайну. Мой же выбор был сделан в сторону среды разработки PHPStorm, с дополнительным расширением Xdebug, которое сокращало время на поиск ошибок в разы, позволяя отлавливать ошибки, через точку останова проходя дебагом.

Кроме того, чтобы в конечно итоге все заработало на локальной машине, необходим также локальный сервер.

Для Фреймворка React Native возможно дополнительно установить компонент Expo.

Expo – это набор инструментов, которые выполняют сборку приложения на локальном порту. Далее при помощи специального ПО для мобильного

устройства, есть возможность в онлайн режиме работать с кодом и одновременно видеть внесенные изменения, в нашем мобильном приложении [22].

Для развертывания back-endи базы данных необходим полноценный сервер, можно рассмотреть наиболее популярные готовые сервера Denwer и OpenServer, а также XAMPP.

Denver – это одна из первых программ для веб-разработчиков, реализованная нашим соотечественником Дмитрием Котеровым. В свое время, denwer был монополистом на рынке, но по истечению времени его сопровождение прекратилось. Главным минус, что очень трудно совместим с новыми операционными системами. Половина документации применима только для старых операционных систем, а также хостинг сайтов хранения документации периодически останавливается. Поддерживает PHP только до версии 5.2 [23].

XAMPP – почти голый локальный сервер, содержащий Apache.Имеет огромное количество дополнительных расширений/библиотек, дающий возможность запускать полноценный сервер. Из-за его простоты, производительность данного сервера преобладает над остальными, но новичку поднять данный сервер без прочтения документации невозможно, но документация актуальна и подробная. Основным недостатком является совместимость только с 32-х разрядными версиями Window, причем последний выпущенный официальный дистрибутив только для Window 7, хотя его и возможно установить на Window 10, но появляется риск снижения отказоустойчивости [24].

OpenServer – по сравнению с другими локальными серверами, данный сервер более автоматизированный, причем весь программный комплекс имеет в разы больше компонентов по его настройке, а также имеет богатый набор вспомогательного программного обеспечения, не требующих инсталляцию.

Сопровождение происходит по сегодняшний день. Поддерживает актуальные операционные системы, актуальные версии языков программирования, имеет невероятно простой и дружелюбный интерфейс. Умеет автоматически настраивать системные настройки компьютера под свои настроечные данные. Основным, но не критичным минусом является его длительность запуска, а также рестарта при изменении конфигурационных файлов [25].

Сборка OpenServer включает в себя:

1. HTTP модули:
   1.1. Apache.
   1.2. Nginx.
2. СУБД модули:
   2.1. MySQL.
   2.2. PostgreSQL.
3. PHP модули:
   3.1. PHP версии 5.x.
   3.2. PHP версии 7.x.
4. Дополнительный набор инструментов:
   4.1. HeidiSQL
   4.2. Adminer
   4.3. PHPMyAdmin
   4.4. PHPPgAdmin
   4.5. PgAdmin
   4.6. Perl
   4.7. FTP сервер
   4.8. Sendmail
   4.9. Memcachedсервер.

Конечно же, выбор очевиден и это будет OpenServer, так как он позволяет более гибко настраивать сервер под себя, под свои нужды. В отличие от Denwer

поддерживает свежие версии PHP, а в отличие от XAMPP поддерживает актуальные операционные системы. Дополнительно OpenServer легче развернуть и поднять, а также имеет большой выбор подключаемых компонентов.

В результате было принято решение использовать:

1. Для реализации интерфейса (front-end):
    1.1. Фреймворк - React Native
    1.2. Среда разработки - Visual Studio Code
2. Для реализации API (back-end):
    2.1. Язык программирования PHP, Фреймворк Yii.
    2.2. Среда разработки – PhpStorm.
3. Для реализации базы данных:
    3.1. СУБД – OpenServer.

1.3 Формализованное описание технического задания

на разработку мобильного приложения

«Мобильное приложение для организации совместных велопоходов»

Составлен на основе ГОСТ 34.602-89 «Техническое задание на создание автоматизированной системы».

1. Общие сведения.

1.1. Название организации-заказчика.

ООО «Бюро Путешествий»

1.2. Название продукта разработки (проектирования).

«Мобильное приложение для организации совместных велопоходов»

1.3. Назначение продукта.

Упрощение организации велопоходов.

1.4.Плановые сроки начала и окончания работ.

В соответствии с планом выполнения ВКР (20.12.2020 – 28.02.2021).

2. Характеристика области применения продукта.

2.1. Процессы и структуры, в которых предполагается использование продукта разработки.

Организация и участие в совместных велопоходах.

2.2. Характеристика персонала (количество, квалификация, степень готовности)

2.2.1. Разработчик – знание SQL, PHP, JavaScript и React Native.

2.2.2. Администратор БД – умение обслуживать базы данные и знание SQL.

3. Требования к продукту разработки.

3.1. Требования к продукту в целом.

3.1.1 ПО должно кроссплатформенным.

3.1.2 Поддерживать разный размер дисплеев.

3.2. Аппаратные требования:

3.2.1. Мобильное устройство, с возможностью выхода в интернет.

3.2.2. Сервер приложения:

Наименование	Описание
Процессор	4 ядра (8 логических потоков), частота – 3-3,5 ГГц и более
Оперативная память	8 Гб и более
Свободное дисковое пространство	50 Гб и более
Пропускная способность сетевого интерфейса	1 Гбит/с

3.2.3. Сервер базы данных:

Наименование	Описание
Тип накопителя	SSD
Процессор	4 ядра (8 логических потоков), частота – 2,5 ГГц и более
Оперативная память	16 Гб и более
Свободное дисковое пространство	300 Гб и более
Пропускная способность сетевого интерфейса	1 Гбит/с

3.3. Указание системного программного обеспечения (операционные системы, браузеры, программные платформы и т.п.).

Любая платформа, поддерживающая современные технологии.

3.4. Указание программного обеспечения, используемого для реализации.

Любое бесплатное программное обеспечение.

3.5. Для сетевых систем – особенности реализации серверной и клиентской частей.

Без ограничений.

3.6. Форматы входных и выходных данных:

Ограничений нет.

3.7. Источники данных и порядок их ввода в систему (программу), порядок вывода, хранения.

Источников входных данных является пользователь и мобильное устройство при помощи, которого выполняется ввод и вывод данных. Хранение данных, их обработка и чтение осуществляется посредством использования базы данных.

3.8. Порядок взаимодействия с другими системами, возможности обмена информацией.

Не предусмотрены.

3.9. Меры защиты информации.

Присутствует аутентификация по логину (номер телефона) и паролю (минимальное количество символов 6). После первичного входа, далее вход может быть осуществлен при помощи присвоенного токена авторизации.

4. Требования к пользовательскому интерфейсу.

4.1. Размер компонентов должен быть читабельным пользователю с нормальным зрением.

4.2. Экрана должен быть визуально разбит на три части: заголовок (при необходимости), основная часть, навигация по приложению.

4.3. Наличие проверок вводимых пользователем данных.

5. Перечень сопроводительной документации:

5.1. Техническая документация расположения файлов части front-endприложения.

5.2. Техническая документация расположения файлов части back-endприложения.

6. Порядок сдачи-приемки продукта.

В соответствии с планом выполнения ВКР.

Глава 2. Разработка мобильного приложения для организации совместных велопоходов.

2.1 Модельные представления основных объектов разработки. Этапы разработки.

Перед началом разработки системы и для упрощения дальнейшей работы составим и рассмотрим некоторые UML диаграммы по разработке и работе будущего приложения.

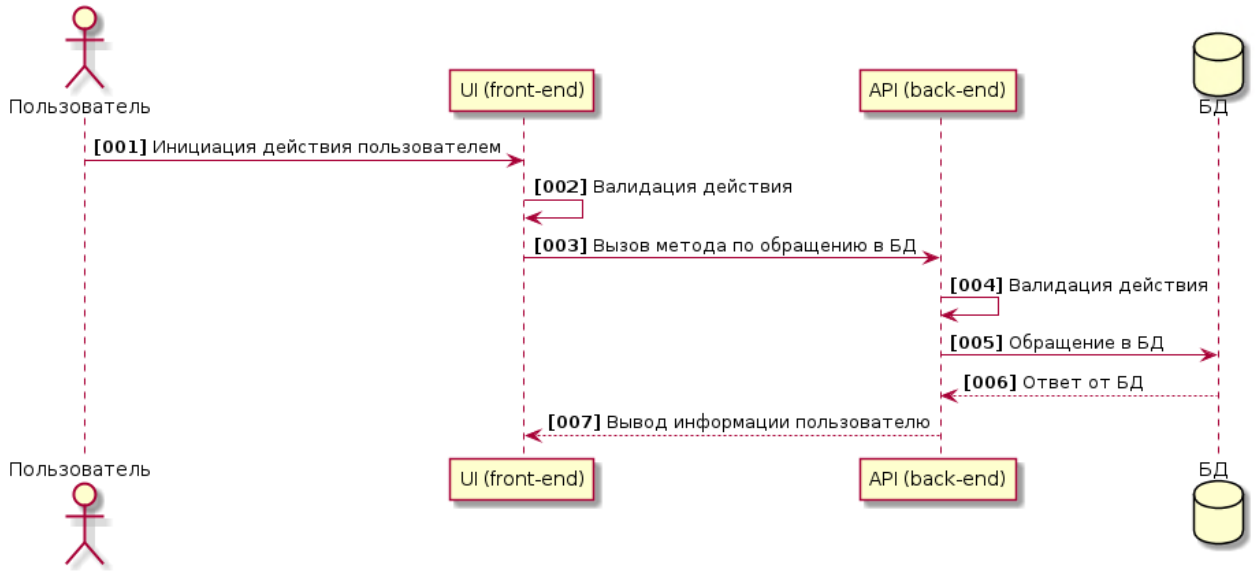

Рисунок 3 - Диаграмма последовательности работы приложения

Как можно наблюдать инициатором всех действий будет являться пользователь, он работает с интерфейсной части приложения, далее фронтовая часть приложения вызывает необходимую функциональность back-end, также при необходимости передавая какие-либо параметры, а бэк уже проводит валидацию, если это требуется и общается с базой данной. Аналогичным образом происходит ответ от всех методов.

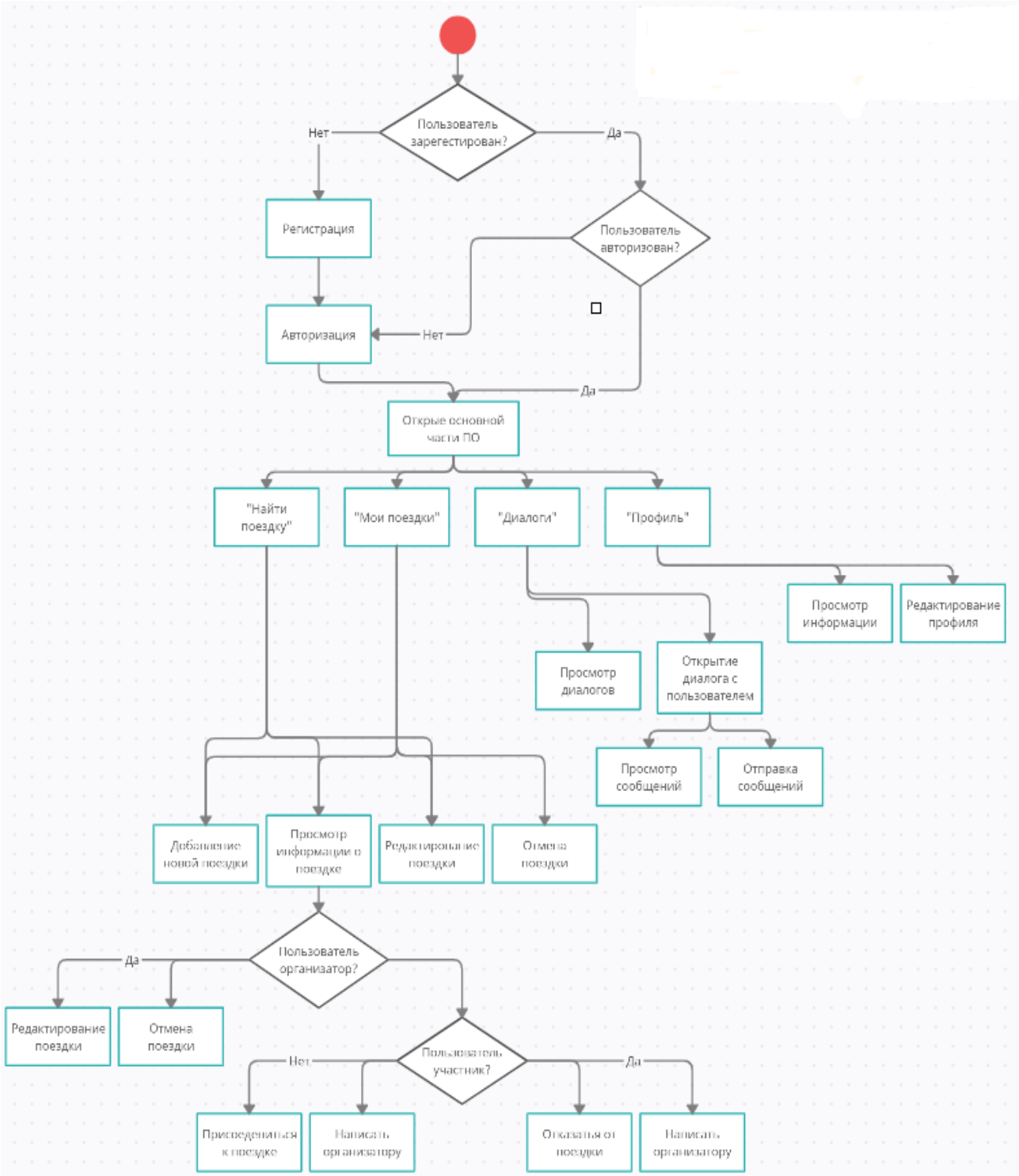

Рисунок 4 – Диаграмма деятельности работы в ПО

Благодаря данному алгоритму, мы определились с основными экранами ПО и с основным функционалом, конечно, данная схема не является конечной и будет развиваться, декомпозироваться на более мелкие.

2.1.1 Разработка базы данных

После разработки дизайна, мы уже понимаем, какая информация у нас будет отображаться в приложении, вследствие чего смело приступаем к разработке базы данных.

Проектирование базы данных осуществлялось при помощи онлайн сервиса DBDesigner (адрес - https://app.dbdesigner.net/designer/schema/391000). Данный сервис бесплатный и самым главным плюсом является, что по окончании можно данный проект экспортировать в виде готово SQL запроса для создания базы данных, что позволяет нам сэкономить время.

Результат проектирования базы данных можно посмотреть на рисунке 16, которая состоит из шести таблиц:

1. User – информация о пользователях.

   Поля (тип данных):

   1.1. Id (integer) – уникальное поле таблицы.

   1.2. Username (varchar (100)) – наименование пользователя.

   1.3. Email (varchar (100)) – электронная почта.

   1.4. Phone (varchar (11)) – номер телефона.

   1.5. Comment (varchar (250))– дополнительная информация о себе.

   1.6. Image (binary) – картинка-аватар.

   1.7. BirthDate (date) – дата рождения.

   1.8. PasswordHash (varchar (max)) – хеш пароля.

   1.9. CreatedAd (datetime) – дата регистрации пользователя.

   1.10. CityId (integer) – населенный пункт.

2. Cities – справочник населенных пунктов.

   Поля (тип данных):

   2.1. Id (integer) – уникальное поле таблицы.

   2.2. Name (varchar (255)) – наименование населенного пункта.

3. Trips – информация о маршрутах.

3.1. Id (integer) – уникальное поле таблицы.

3.2. UserId (integer) – ид пользователя, являющегося организатором.

3.3. FromCityId (integer) – населенный пункт отправления.

3.4. ToCityId (integer) – населенный пункт прибытия.

3.5. Comment (varchar (max)) – дополнительная информация.

3.6. PlaceMeet (varchar (255)) – информация о месте встречи.

3.7. StartDateTime (datetime) – дата и время начала.

3.8. Deleted (boolean) – признак актуальности.

4. Bookings – информация об участниках.

   Поля (тип данных):

4.1. Id (integer) – уникальное поле таблицы.

4.2. UserID (integer) – ид пользователя.

4.3. TripId (integer) – ид маршрута.

5. Dialogues – диалоги пользователей.

   Поля (тип данных):

5.1. Id (integer) – уникальное поле таблицы.

5.2. Code (varchar (max)) – уникальный код диалога.

5.3. SenderId (integer) – ид пользователя отправителя.

5.4. ReceiverId (integer) – ид пользователя получателя.

5.5. LastMessage (varchar (max)) – последнее отправленное сообщение.

5.6. LastMessageUserId (integer) – ид пользователя, отправившего последнее сообщение.

5.7. Created (datetime) – дата создания диалога.

5.8. Updated (datetime) – дата последнего отправленного сообщения.

6. Messages – сообщения диалогов.

   Поля (тип данных):

6.1. Id (integer) – уникальное поле таблицы.

6.2. UserId (integer) – ид пользователя.

6.3. DialogueId (integer) – ид диалога.

6.4. Text (varchar (max)) – текст сообщения.

6.5. Created (datetime) – время отправки сообщения.

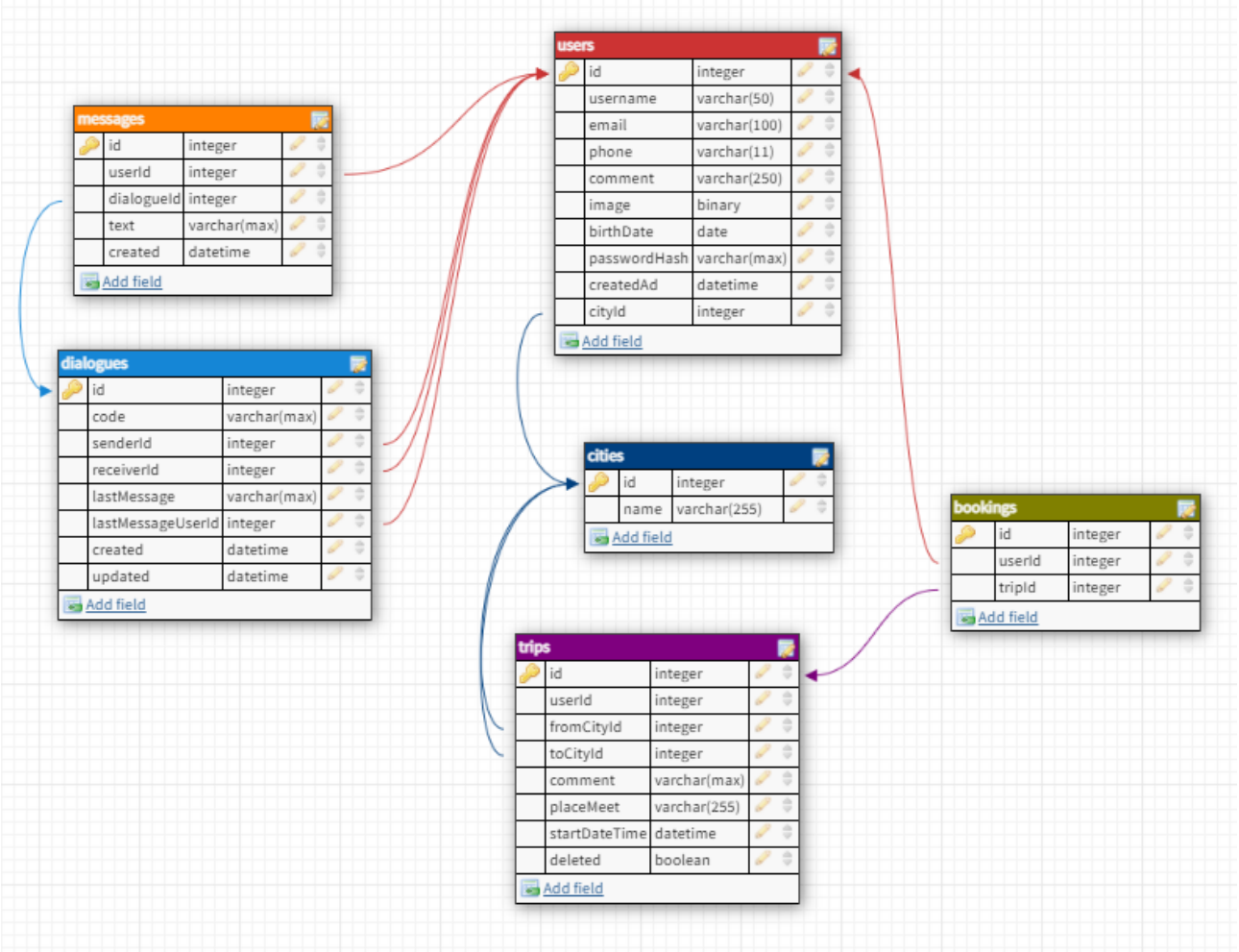

Рисунок 5 – Проект базы данных

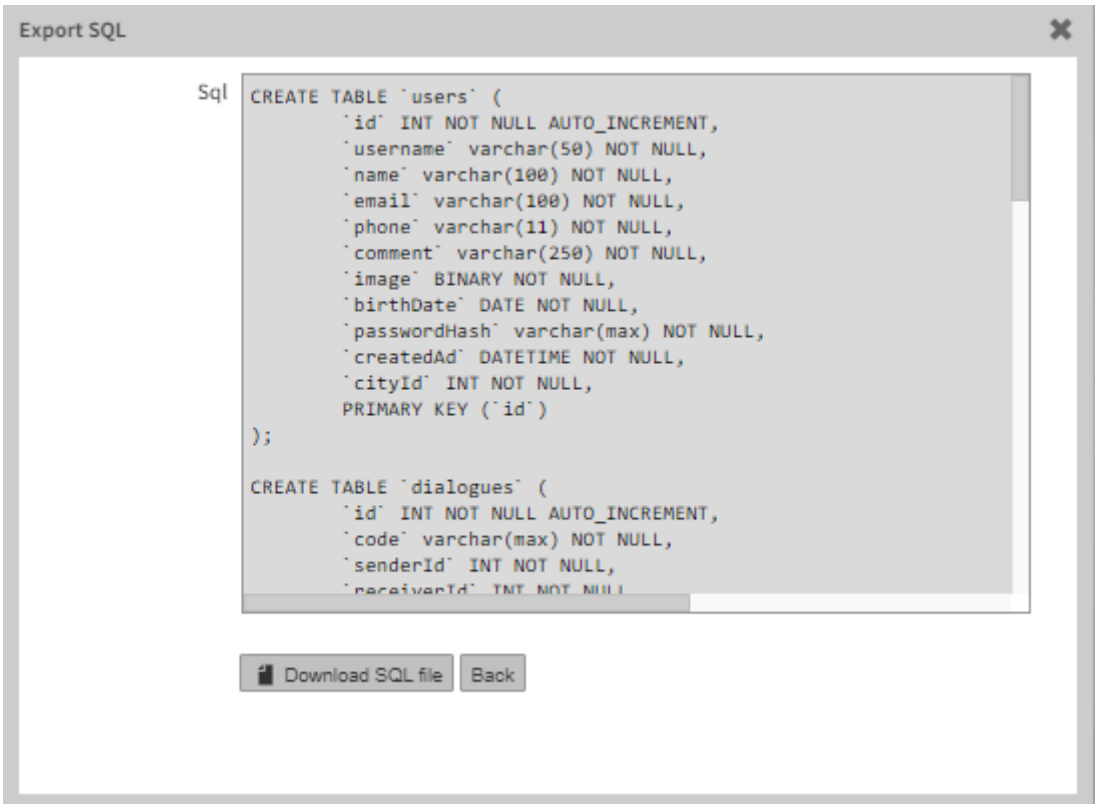

Рисунок 6 – Сгенерированный SQLзапрос по созданию таблиц в БД

После генерации SQLзапроса, его необходимо доработать, добавив запрос на создание базы данных. Соответственно остается только выполнить данный запрос на нашем локальном сервере и появится физическая готовая база данных, которую остается только наполнять данными.

2.1.2 Разработка API

В соответствии с принятым решением, взаимодействие с базой данных, реализовывать будем при помощи API, каждый метод, которого будет иметь свою уникальную задачу, а обращение к ним будет путем вызова из интерфейсной части приложения или из другого метода.

Для разработки API приложения используем Фреймворк «Yii» (Easy efficient extensible) предназначенный для быстрой разработки веб-приложений. В качестве среды разработки будет PHPStrom. Дистрибутив программного обеспечения PHPStrom, можно взять с официального сайта https://www.jetbrains.com/ru-ru/phpstorm.

Для установки Фреймворка Yii необходимо установить Composer. Скачать его можно с официального сайта https://getcomposer.org/download. Далее согласно официальной документации Фреймворка Yii рекомендуется установить имеющий шаблон, вместо создания проекта с нуля, выполним команду «composer create-project --prefer-dist yiisoft/yii2-app-basic basic».

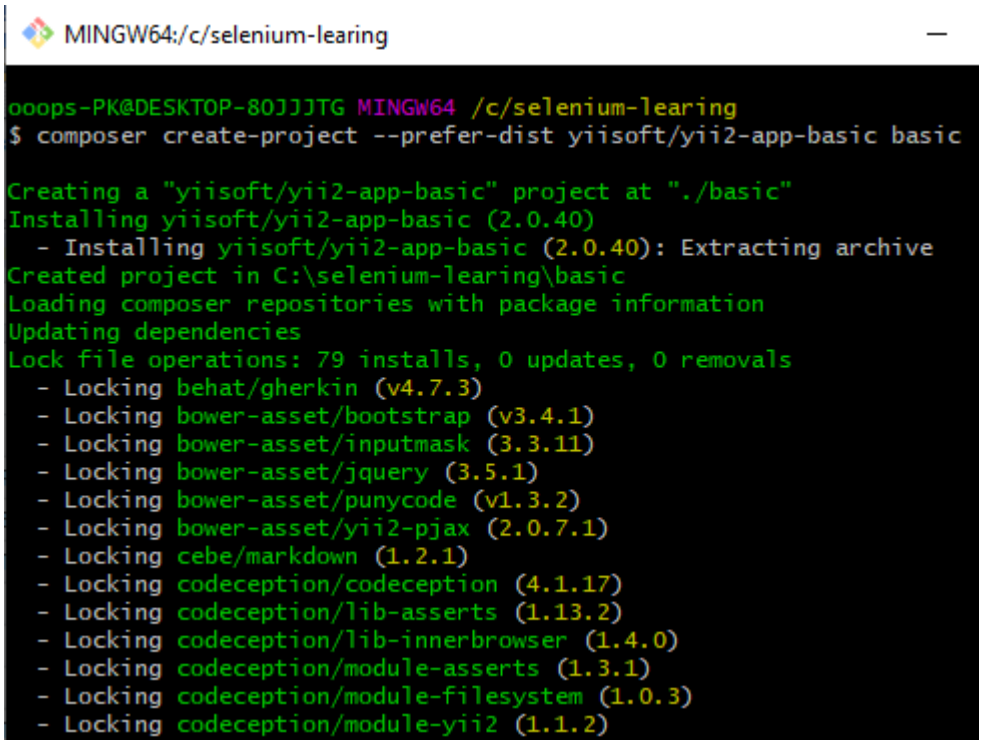

Рисунок 7 – создание проекта на основе Фреймворка Yii

Так как это шаблон проекта веб-сайта, в нем присутствует огромное количество файлов, не нужных нам. Удаляем все файлы, оставляя только папки: vendor, web, config и файлы composer.json, composer.lock. Теперь мы имеем голый проект.

Дополнительно, для корректной работы сервера (согласно официальной документации https://www.yiiframework.com/doc/guide/2.0/ru/start-installation), необходимо настроить конфигурацию Nginx (http–сервер, принимающий запросы от пользователя), который, как уже писалось ранее, встроен в «openServer».

После произведенных всех настроек, создаем подключение к базе данных. Для удобства параметры подключения к базе данных вынесем в файл «.env», который создадим в корне проекта, а далее будем вызывать через функцию «getenv(`наименование параметра`)».

Рисунок 8 – подключение к базе данных

Затем дополнительно создадим два файла конфигурации «request.php» и «response.php». В данных файлах, необходимо настроить обработку формы запросы и ответа. Для этого в «request.php» включим синтаксический анализ входного JSON (потребуется при вызове метода POST), а в файле «response.php» реализуем формат возвращаемых данных также в формате JSON.

request.php

```php
<?php

return [
    'baseUrl' => '/api',
    'parsers' => [
        'application/json' => 'yii\web\JsonParser'
    ]
];
```

Рисунок 9 – код файла «request.php»

response.php

```php
<?php

return[
    'format' => yii\web\Response::FORMAT_JSON
];
```

Рисунок 10 – код файла «response.php»

Как можно наблюдать на рисунке 10, дополнительно был добавлен параметр «baseUrl», который добавится в адрес методов (параметр не обязательный, делается для наглядности адреса API).

На следующем шаге перейдем к созданию метода по получению всех населенных пунктов из таблицы «cities» ранее созданной базы данных. Для этого в папке «app» создадим папку «modules». В папке «modules» создадим папку «v1», означающую первую версию методов. Использование версий методов является рекомендуемым, так как в будущем при доработке методов вместо изменения первой версии, лучше добавить вторую версию, и тогда появляется уверенность, что первая версия будет продолжать работать корректно.

В папке «v1» создадим еще подпапки «controllers», «repositories», «useCase». Также здесь создадим файл с пустым классом «module.php», необходимый, чтобы наследовать тип встроенного функционального класса.

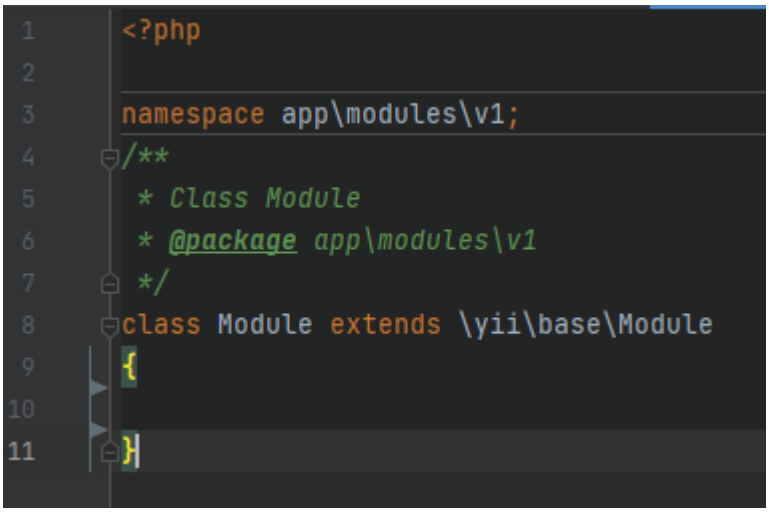

Рисунок 11 – код файла «module.php»

Переходим к реализации метода по получению данных из базы данных. Начинаем с создания SQL запроса по поиску информации о населенных пунктах в базе данных. Для этого в папке «repositories» создадим файл «CitiesRepository.php». После объявим в нем класс и при помощи dbQuery преобразуем стандартный sql запрос по выборке всех данных «SELECT id, name FROM cities».

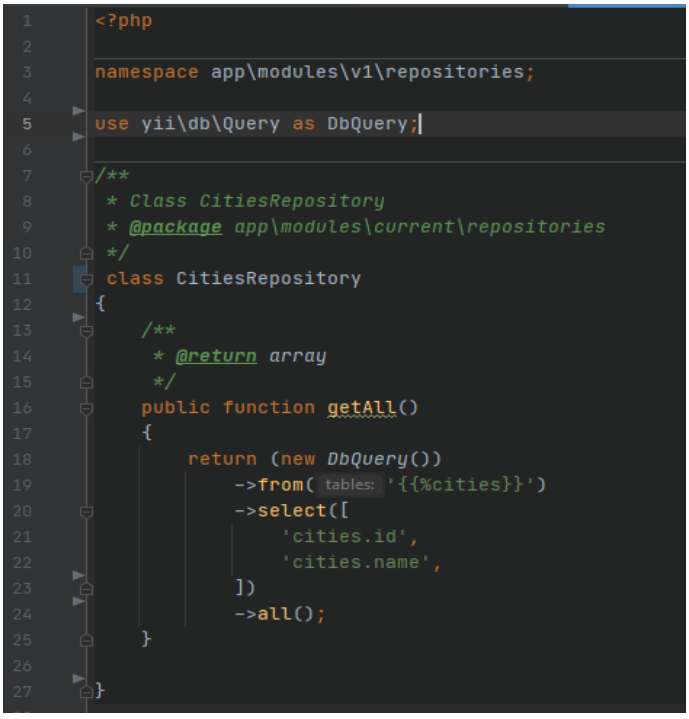

Рисунок 12 – поиск населенных пунктов из базы данных

После того, как был написан метод по получению данных, переходим к написанию логики запроса и обработки ответа. Добавим файлы в папке «useCase»: «ListCitiesResponse.php» (в данном файле просто указана функция конструктора, для возвращаемого результата) и «ListCitiesUseCase.php», в котором будет также указан конструктор и вызов ранее написанного метода по получению информации по населенным пунктам.

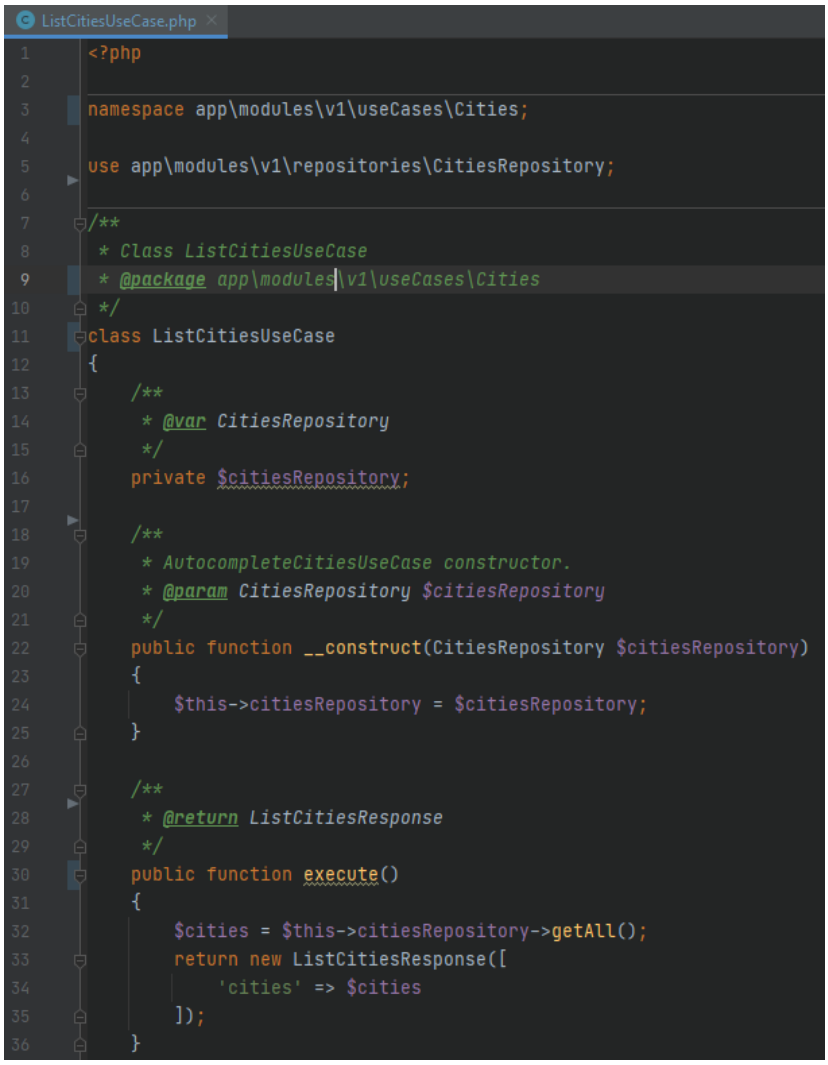

Рисунок 13 – код файла «ListCitiesUseCase.php»

Затем в папке «controllers» добавим файл «CitiesController.php». В данном файле зашита общая логика по принятию запроса, его обработке и возврату результата. Так как входные параметры в данном методе отсутствуют, то обрабатывать будет только результат.

CitiesController.php

```php
<?php

namespace app\modules\v1\controllers;

use ...

/**
 * Class CitiesController
 * @package app\modules\v1\controllers
 */
class CitiesController extends \yii\web\Controller
{

    /**
     * @return array
     */
    public function actionList()
    {
        $useCase = new ListCitiesUseCase(new CitiesRepository());

        $response = $useCase->execute();
            return [
                'success' => true,
                'data' => [
                    'cities' => $response->cities,
                ],
            ];

    }
}
```

Рисунок 14 – основная логика обработки метода по получению населенных пунктов

Как можно увидеть на рисунке 14 у нас изначально вызывается инициализация свойств, для создаваемого объекта $useCase. После вызываем метод execute, в ранее описанном файле «ListCitiesUseCase.php», который в сою очередь вызывает метод из репозитория по получению информации непосредственно из базы данных. Данная хитрая логика нужна, для корректного преобразования возвращаемого ответа и легкости дальнейшего сопровождения, таким образом, это позволит избежать дублирования кода.

После написания метода, остается только присвоить ему адрес. Для этого вернемся в папку «config» и добавим файл «url-manager.php».

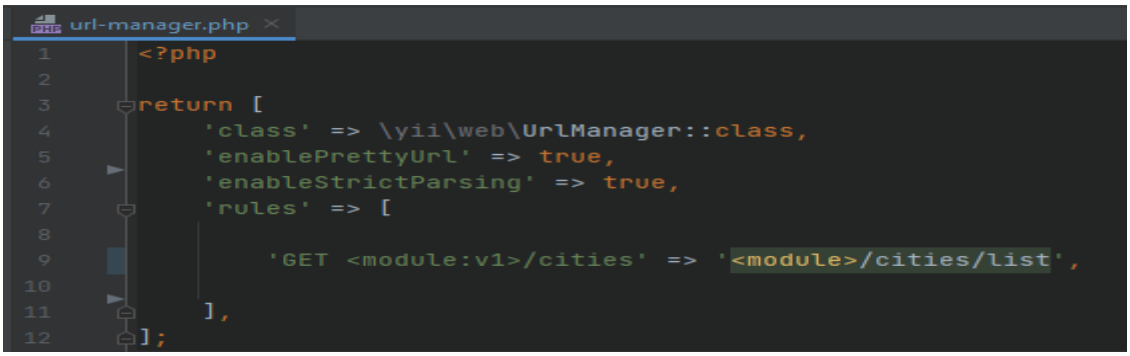

Рисунок 15 – url адреса методов

Обязательно указываем дополнительно два параметра: «enablePrettyUrl» (включает использование красивых url адресов) и «enableStrictParsing» (включает строгий синтаксический анализ проверки url адресов).

В связи с тем, что мы используем поддержку версионности методов, в url указываем тег module и версию метода, а также добавляем файл «modules.php», в котором указываем полный адрес соответствия версий и методов.

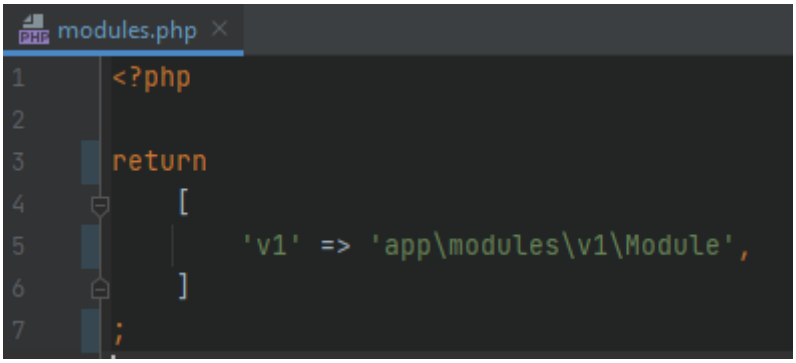

Рисунок 16 – настройка адреса версионности методов

2.1.3 Разработка интерфейса

На данном этапе приступаем к разработке интерфейсной части мобильного приложения.

Первым шагом, необходимо установить программные обеспечения, рассмотренные в пункте 1.2, которые необходимы для программирования, то есть установим Visual Studio Code. Дистрибутив программного обеспечения Visual Studio Code, можно взять с официального сайта https://code.visualstudio.com.

После установки Visual Studio Code, приступаем к созданию проекта. Для этого необходимо установить набор инструментов Expo, выполнив через командную строку следующее «npm install --global expo-cli».

Далее, продолжая работу в командной строке, выполним команду по созданию пустого проекта «expo init FrontEnd --template=blank».

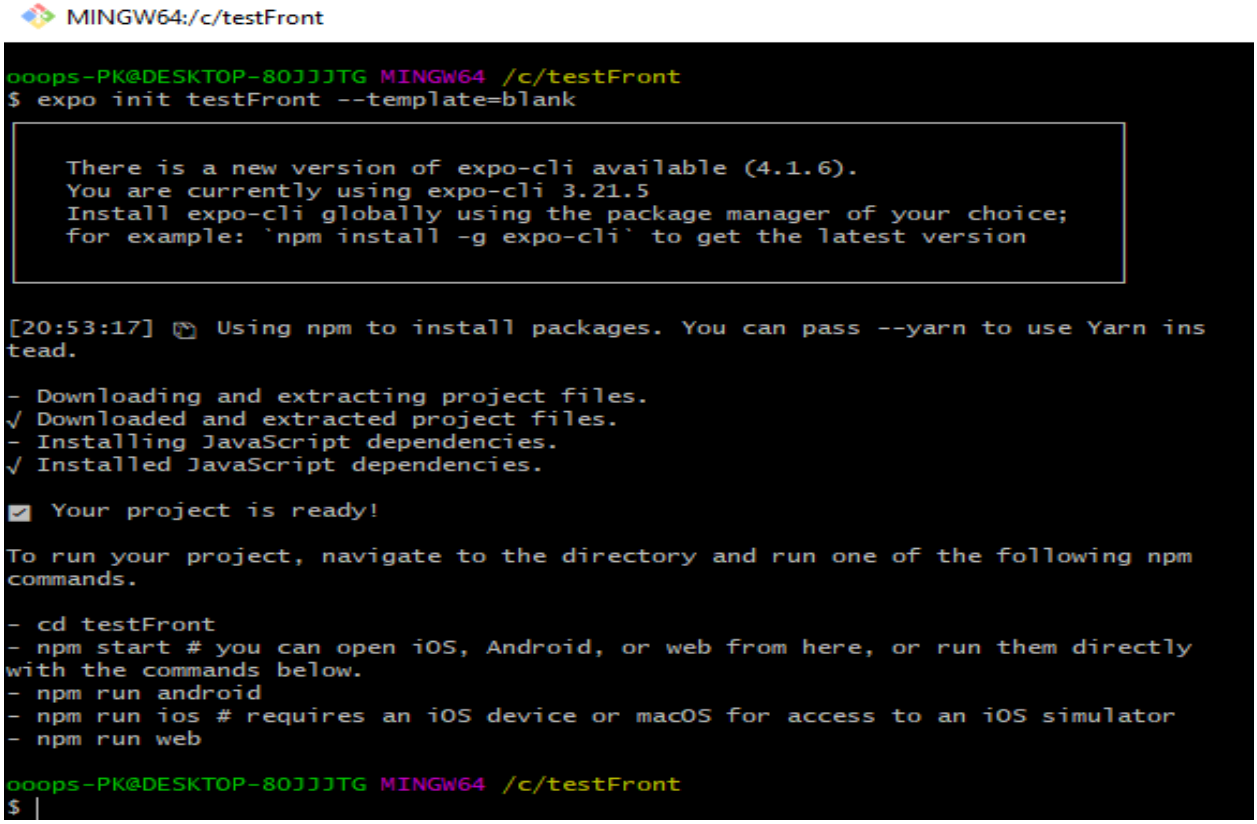

Рисунок 17 – создание пустого шаблона проекта

Соответственно после выполнения данной команды, у нас создастся проект мобильного приложения, с пустым шаблоном файлов, который уже сейчас можно запустить и протестировать на телефоне.

Следующим шагом, необходимо созданный проект открыть с помощью Visual Studio Code, нажатием сочетания клавиш «Ctrrl + O», после выбрав папку, ранее созданного проекта.

Далее создадим папку «screens» в нашем проекте, которая будет хранить в себе все экраны приложения. В ней создадим файл «MyProfile.js» предназначенный, для отображения информации профиля пользователя.

Важно помнить, что разрабатывать экраны приложения будем через классы, а не через стрелочные функции, так как при использовании асинхронных запросов к api, производительно будет лучше. Поэтому создаем шаблон экрана, согласно рисунку 18.

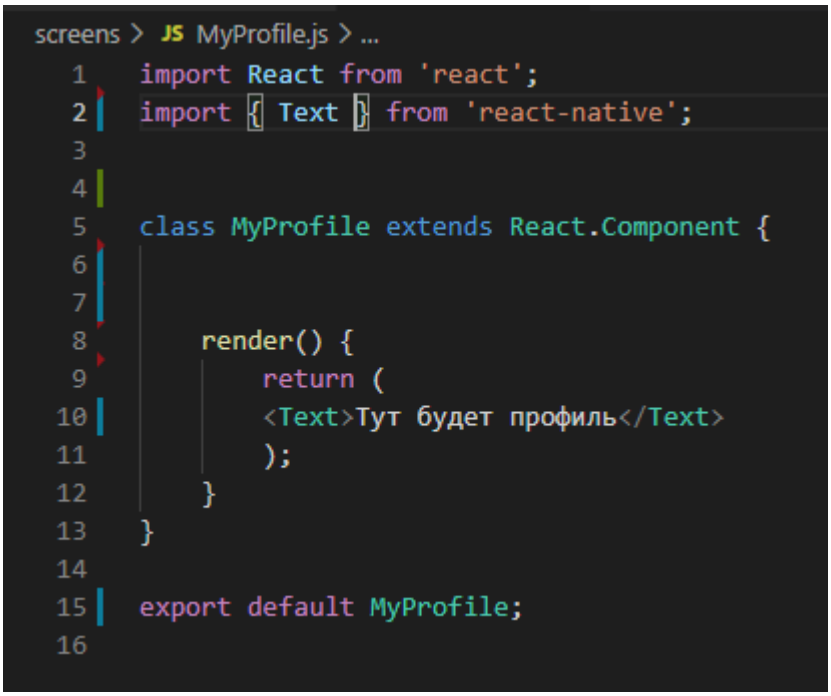

Рисунок 18 – шаблон экрана профиля пользователя

Подобным образом создадим еще три основных экрана («Мои маршруты», «Найти маршрут», «Диалоги») пользователя, меняя только наименование и выводимый текст, для визуального различия.

После того, как будут созданы основные экраны приложения, переходим к созданию навигации в мобильном приложении. Использовать будем официальную библиотеку React Navigation (документация - https://reactnavigation.org/docs). Для установки нижней навигационной панели необходимо выполнить внутри нашего проекта команду «npm install @react-navigation/bottom-tabs», в командной строке (командная строка вызывается в Visual Studio Code сочетанием клавиш «Ctrl+`»). Также установим «npm install @react-navigation/stack», предназначенный для компоновки нескольких экранов в один путь («хлебные крошки»).

Так как в нашем приложении будет не один экран, а несколько, для удобства настройку навигаторов приложения вынесем в отдельную папку проекта «Navigators».

Рисунок 19 – Код навигационной панели

Далее, добавляем файл «appNavigator.js» в папку «Navigators». В данном файле, будет лежать настройка по работе и стилистике нижней навигационной панели приложения. Согласно рисунку 4, в основной части приложения у нас будет четыре экрана, для которых мы ранее создали пустые шаблоны, поэтому в навигационной панели у нас также будет четыре кнопки. Здесь же в настройках меню, делаем условие, что если экран выбран, то кнопку оформляем одном стиле, если не выбран, то в другом.

После, сразу сделаем шаблон стэк-навигатора для созданных экранов, то есть в данной же папке добавим файлы: «myTripNavigator.js» (включает экран моих маршрутов и мною выбранных), «tripNavigator.js» (включает экран всех маршрутов), «dialoguesNavigator.js» (включает экран диалогов), «profileNavigator.js» (включает экран профиля пользователя). А после эти стэк-навигаторы включим в файл «appNavigator.js».

navigators > JS profileNavigator.js > ...

```
1   import React from 'react';
2   import {createStackNavigator} from 'react-navigation-stack';
3   import MyProfile from '../screens/MyProfile';
4
5   export const profileNavigator = createStackNavigator(
6       {
7           Profile: MyProfile,
8       },
9       {
10          initialRouteName: 'Profile',
11          defaultNavigationOptions: {
12              headerShown: false,
13          }
14      },
15  );
```

Рисунок 20 – код «profileNavigator.js»

То есть в дальнейшем, при создании новых экранов, они будут добавлять в соответствующие стэк-навигаторы, например, в стэк-навигатора «profileNavigator», добавится экран «Редактирование профиля». И при нажатии на кнопку назад, находясь на экране «Редактирование профиля», открываться

будет всегда экран «Профиль». В стэк-навигаторе «dialoguesNavigator», добавится экран «Диалог с пользователем», при попытке возвратиться назад, будет всегда открываться экран «Диалоги». По аналогии работают остальные стэк-навигаторы.

Далее требуется изменить файл «app.js». Этот файл инициирует приложение и выводит компоненты в заданной ему последовательности. Дополнительно помним, что в данном файле не должно быть указано логики. То есть в данном файле, необходимо сделать, чтобы он выводил пользователю ранее добавленный нами файл «appNavigator.js», который вызовет другой стэк-навигатора, в котором будет указан конкретный экран приложения.

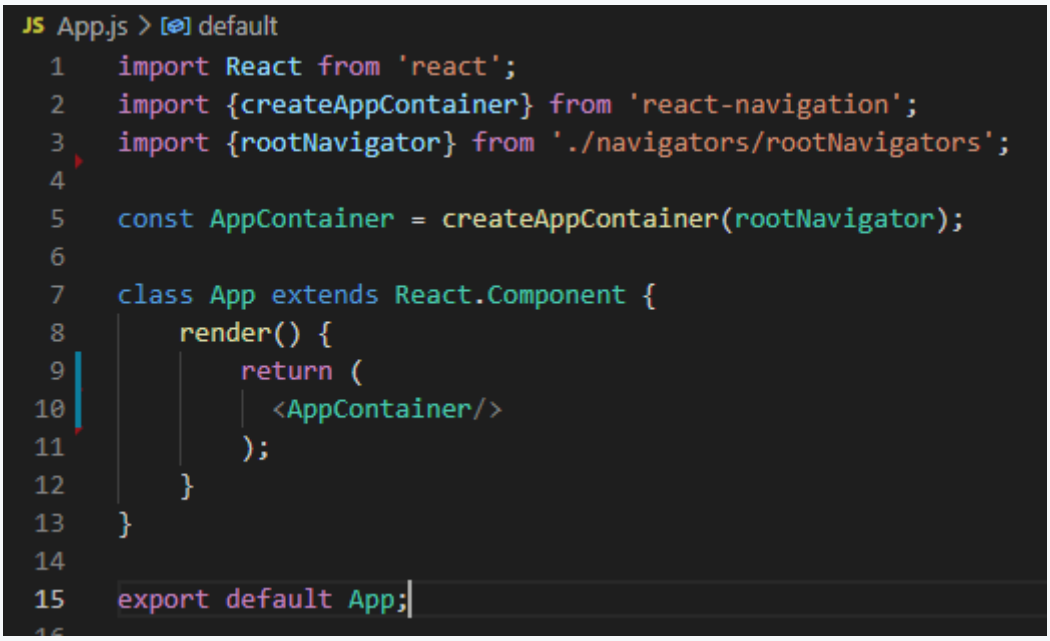

Рисунок 21 – код файла «app.js»

После подготовки основы приложения, нам необходимо реализовать авторизацию пользователя и соответственно его регистрацию. Поэтому по аналогии с прошлым опытом, добавляем два новых экрана.

На экране «Авторизация» добавляем два поля для ввода номера телефона и пароля, а также две кнопки «Войти» и «Регистрация». Соответственно при нажатии на кнопку «Регистрация» должен открываться одноименный экран, а при нажатии на кнопку «Войти» должен вызываться метод по работе с базой

данной «auth/login». В случае успешной авторизации открывается основная часть приложение, в противном случае выводятся ошибки.

Переход между экранами выполняем при помощи встроенной функции onPress, указывая в ней функцию навигатора «this.props.navigation.navigate(«AuthRegister»)» (в скобках указывается наименование экрана).

После реализации авторизации и регистрации, теперь при открытии мобильного приложения, нам требуется четкое понимание, авторизован ли пользователь, для понимания какой экран выводить пользователю. Если пользователь не авторизован, то выводим экран «Авторизации», иначе открываем основную часть приложения.

Поэтому, добавим еще один экран с наименованием «InitialScreen.js», в котором будем проверять, есть ли у пользователя токен (который хранится в памяти телефона и присваивается ему в момент авторизации) или нет. Если токен отсутствует или некорректный, то открывает стэк-навигатор авторизации, в противном случае открываем стэк-навигатор основной части приложения.

```
async initialize() {

    try {
        let authToken = !!this.props.authToken;
        let currentUser = {};

        if (authToken) {
            global.defaultFetchHeaders['Authorization'] = 'Bearer ' + this.props.authToken;
            currentUser = this.props.authToken ? await this.getAccountCredentials() : {};
        }
        const screen = currentUser.id ? 'App' : 'Auth';
        setTimeout(() => this.props.navigation.replace(screen), 500);
        await this.props.citiesActions.getCities();
    } catch(error) {
        this.props.navigation.replace('Auth');
    }
}
```

Рисунок 22 – проверка наличия токена пользователя

При инициализации мобильного приложения, также дополнительно в асинхронном режиме необходимо запрашивать справочник городов, при

помощи вызова метода «/cities». Для получения данных будем использовать встроенную функцию в React «fetch», которая может отправлять сетевые запросы на сервер и подгружать необходимую новую информацию по мере необходимости. Также для извлечения данных используем функцию Redux, которая позволяет управлять состоянием приложения.

Для этого создадим новую папку «Actions» и в ней создадим файл «cities.js». Далее создадим папку «Config», создадим два файла: «api.js» и «apiUrls.js». В «api.js» укажем две константы, первая – версия апи, вторая – адрес обращения к api. В файле «apiUrls.js» будем складировать адреса методов по работе в базой данной, в данном случаей добавим адрес метода для получения справочников населенных пунктов.

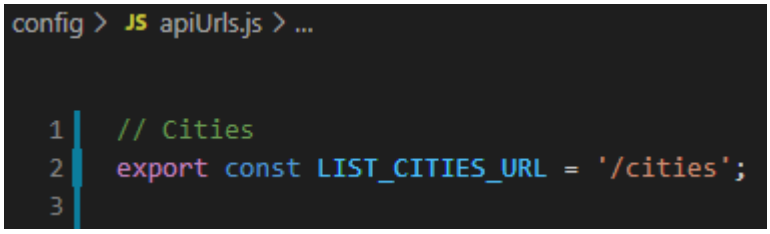

Рисунок 23 – метод по выдаче справочника населенных пунктов

После того, как были указаны переменные по обращению к необходимому методу, возвращаем в файл «cities.js» и разрабатываем вызов метода при помощи строенной функции «fetch» », которая может отправлять сетевые запросы на сервер.

Рисунок 24 – код действия по получению информации из базы данных

Далее, чтобы работать с полученной информацией в любом месте воспользуемся о ранее сказанном Redux. Поэтому добавим папку «reducer», а ней файл «citiesReducer.js».

В данном файле, будем получать информацию и записывать в состояние полученное информацию из какого-то действия, в нашем примере из действия getCities (файл «cities.js»). В последующем, если эта информация нужно, мы на любом экране можно с легкость ее получить, не вызывая повторно метод по работе с базой данной.

```
reducer > JS citiesReducer.js > ...
1   import {LIST_CITIES_SUCCESS} from '../config/actionTypes';
2
3   const initialState = {
4       cities: null,
5   };
6
7   export default function (state = initialState, action) {
8       switch (action.type) {
9           case LIST_CITIES_SUCCESS: {
10              return {...state, cities: action.payload.cities};
11          }
12          default: {
13              return state;
14          }
15      }
16  }
```

Рисунок 25 – получения состояния наименования населенных пунктов

То есть алгоритм работы по обращению с базой данной следующий:

1. Загружаем экран.
2. Вызываем действие из папки «Actions», завязанное на определенный метод по общению с базой данных.
3. Передаем полученные данные «Redux» - записываем в определенное состояние.
4. При необходимости выводим на экран требуемые данные.

Вернемся на экран профиля пользователя. Вместо ранее созданного шаблона, добавим поля по выводу информации из базы данных, таких как наименование пользователя, дату рождения, электронную почту, номер телефона, а также получение изображения пользователя. Далее для заполнения полей профиля пользователя, необходимо обратиться к методу «user/:id». Для этого по аналогии с действием getCities добавляем действие getUser, а также новый файл Redux.

После в файле экрана профиля пользователя необходимо добавить компонент «componentDidMount», который вызывается сразу после рендера экрана. В данном компоненте указать вызов ранее добавленного действия

getUser, для получения данных о пользователе, а также добавить вызов состояния, куда действие getUser запишет полученные данные. И следовательно для вывода информации на экран, достаточно ее получить из состояния командой «this.props.’наименование состояния’.‘наименование поля’».

```
class MyProfile extends React.Component {

    componentDidMount() {
        await this.props.accountActions.getUser({ id: this.props.currentUser.id });
    }

    render() {
        return (
            <View style={stylesCommon.safeContainer} forceInset={{ bottom: 'never' }}>
                <ScrollView showsVerticalScrollIndicator={false} scrollIndicatorInsets={{ right: 1 }} re
                        <View>
                            <View style={styles.container}>
                                <GroupItem>
                                    <FiledFullName>{this.props.user.user.name}</FiledFullName>
                                </GroupItem>
                            </View>
                            <View style={{ flex: 1 }}>
                                <GroupTextView>
                                        <GroupText>
                                            <GrayText>Дата рождения</GrayText>
                                            <Text style={{ fontSize: 18 }}>{getGenitiveDate(this.props.u
                                        </GroupText>
                                        <GroupText>
                                            <GrayText>Электронная почта</GrayText>
                                            <Text style={{ fontSize: 18 }}>{this.props.user.user.email}<
                                        </GroupText>
                                </GroupTextView>
                            </View>
                        </View>
                </ScrollView>
            </View>
        );
    }
}

const mapStateToProps = (state) => ({
    user: state.user.view.user
});
const mapDispatchToProps = (dispatch) => ({
    accountActions: bindActionCreators(accountActions, dispatch),
});
export default connect(mapStateToProps, mapDispatchToProps)(MyProfile);
```

Рисунок 26 – код страницы профиля пользователя

По такому же принципу реализуются остальные экраны приложения.

2.1.4 Хранение исходных данных

Для хранения и контроля версионности проекта, используем Git. Также при помощи контроля версий, в критические моменты, возможно, полностью откатить проект до предыдущего состояния в одну команду.

Каждый день вечера, в которых производилась разработка, фиксируем изменения при помощи команды git commit, а затем командой git push производим закачку внесенных изменений проекта на удаленный репозиторий.

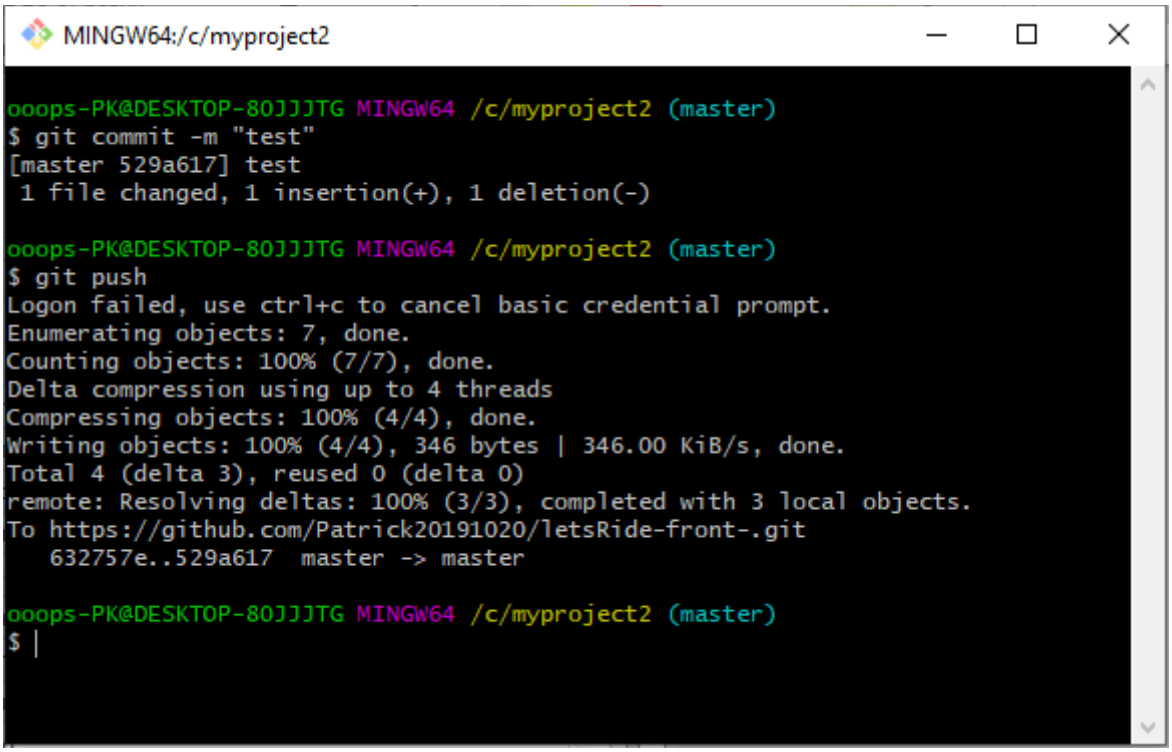

Рисунок 27 – фиксация внесенных изменений в проект на удаленный репозиторий.

Реализация остальных методов выполняется по аналогии с методом по поиску населенных пунктов, при необходимости добавляет валидация, проверка авторизации пользователя и т.д.

2.1.5 Тестирование

Тестирование методов обращения к базе данных осуществлялось при помощи программного обеспечения Postman. Отдельно был протестирован вызов каждого запроса, проверена на корректность структура запроса и ответа. Дополнительно в этот момент проверялась нагрузка на базу данных и наличие блокирующих запросов.

Результаты:

1. Тестирование завершено успешно, обработано успешно 100% запросов.
2. Блокировок на базу данных во время теста не зафиксировано.
3. Утилизация аппаратных ресурсов в пределах нормы.

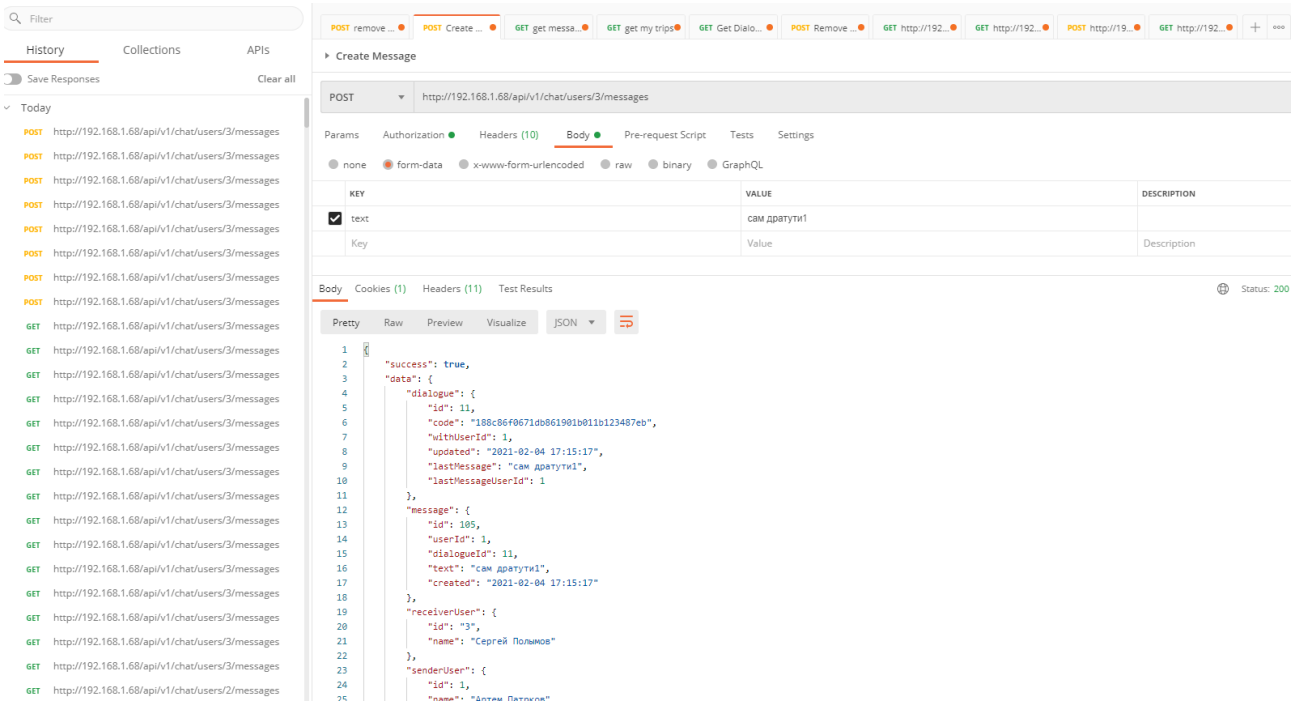

Рисунок 28 – пример запрос и ответа в ПО Postman

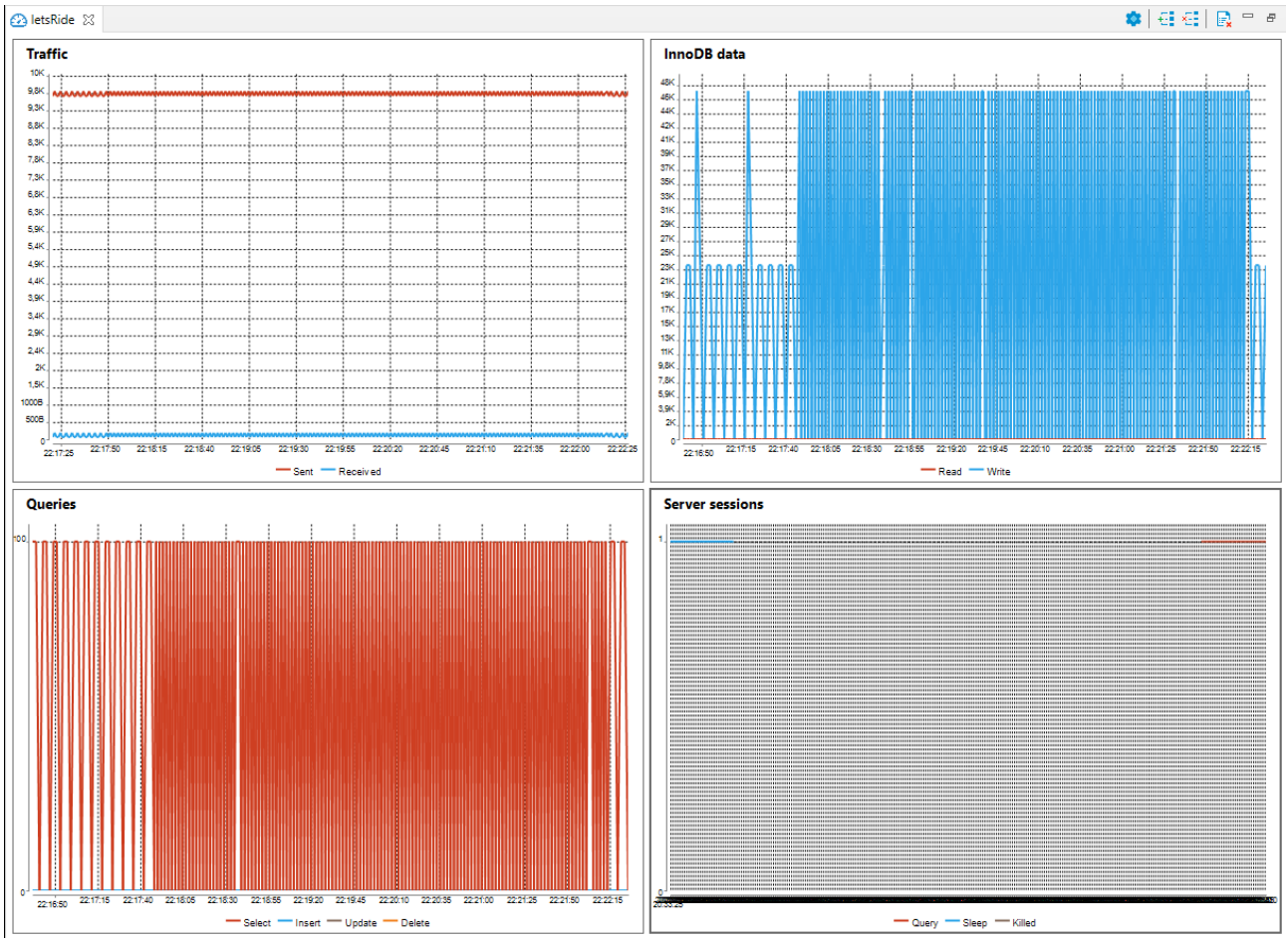

Рисунок 29 – статистика нагрузки на базу данных

	Id	User	Host	db	Command	Time	State	Info
1	62	root	localhost:57169	[NULL]	Sleep	1		[NULL]
2	280	root	localhost:59765	letsride	Query	0	checking query cache for query	/* ApplicationName=DBeaver 7.3.2 - SQLEditor <Script-10.sql> */ show processlist

Рисунок 30 – блокировки базы данных

2.2 Описание продукта

Первый экран, встречающий нового пользователя, это экран авторизации.

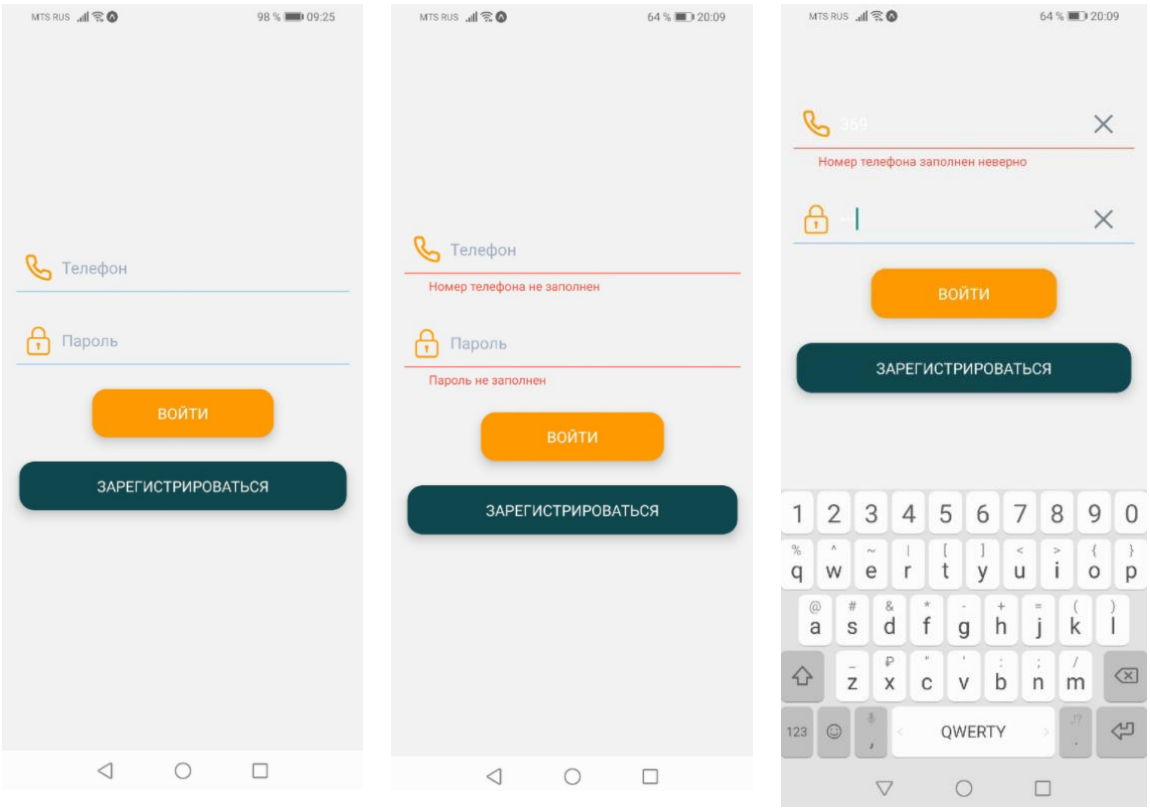

Рисунок 31 – экран авторизации и ошибки

Дополнительно настроена валидация входных данных, при нажатии кнопки «Войти». Каждое вводимое поле имеет свой тип клавиатуры, исходя из необходимой вводимой информацией, например для поля «Телефон» открывает клавиатуры исключительно с цифрами «phone-pad», для ввода пароля стандартная клавиатура. Данное правило предусмотрено для всех подобных полей.

Так как пользователь новый, то необходимо нажать кнопку «Зарегистрироваться» и перейти на экран регистрации пользователя.

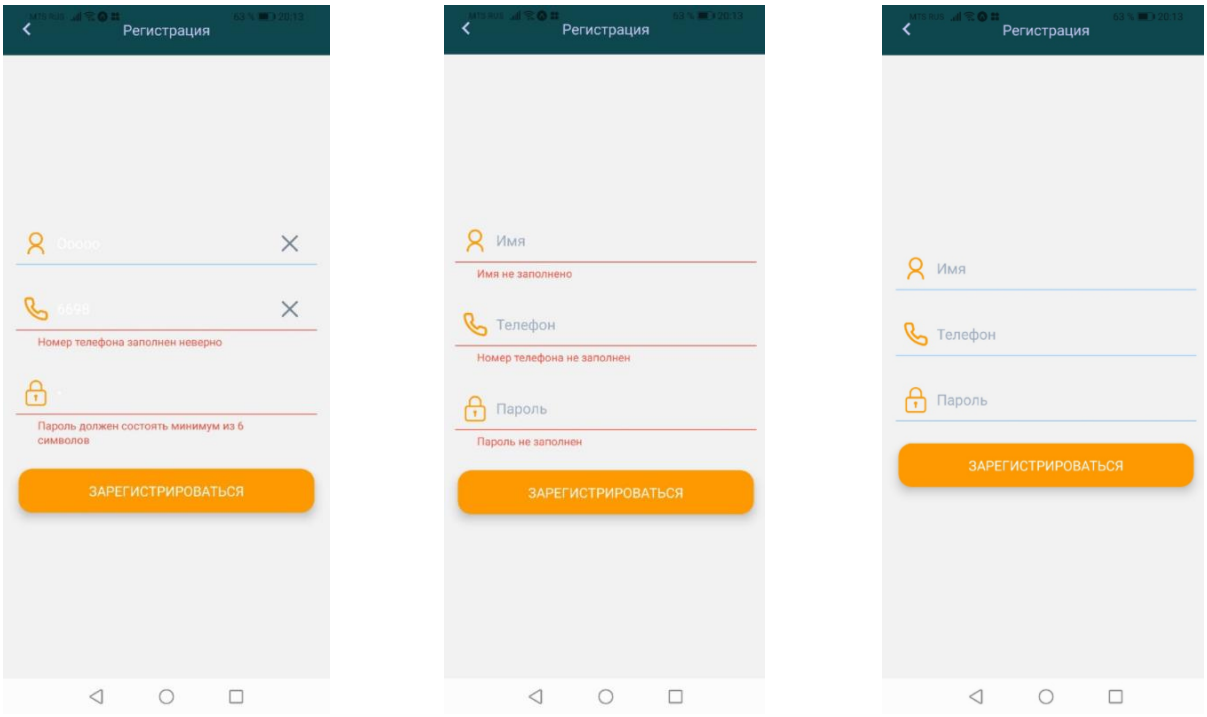

Рисунок 32 – экран регистрации пользователей и ошибки.

Отличительной особенность можно наблюдать, что появился заголовок и кнопка возврата на предыдущий экран. После нажатия на кнопку «Зарегистрироваться» и успешного выполнения регистрации, автоматически перекинет на экран авторизации для входа.

После прохождения успешной авторизации, будет по умолчанию экран профиля текущего пользователя.

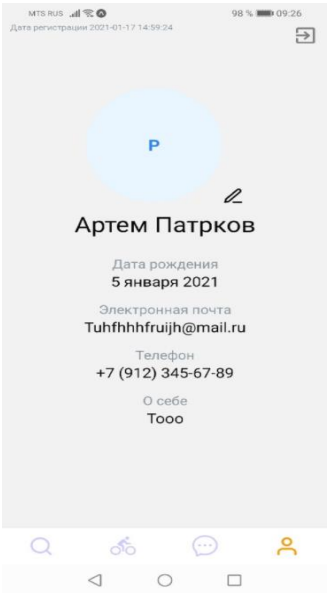

Рисунок 33 – экран профиля пользователя

На данном экране будет указана основная информация о пользователя, которую он внес при регистрации. Также здесь присутствует кнопка выхода из учетной записи и кнопка редактирования профиля. При выходе перекинет на экран авторизации, а при нажатии на кнопку редактирования, на экран редактирования пользователя.

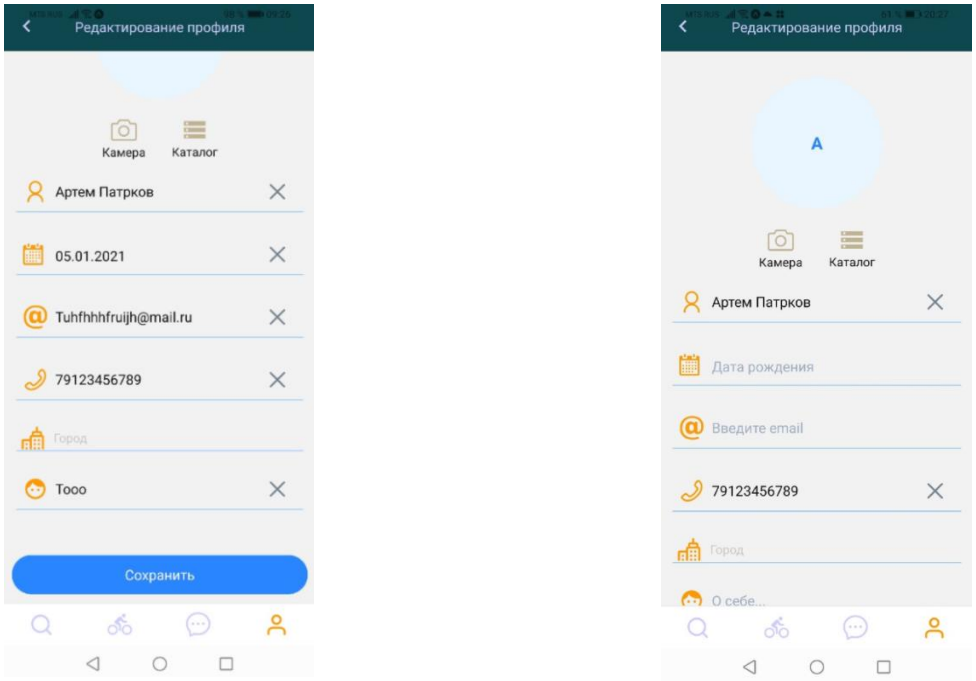

Рисунок 34 – редактирование профиля

На экране возможно заполнить или изменить текущую информацию о пользователе, а также добавить картинку, сделать это возможно двумя путями, либо сразу открыть камеру и сфотографироваться, либо открыть каталог фотографий телефона и загрузить фото. Вся внесенная информация сохраняется, при нажатии на кнопку «Сохранить».

Так как не всегда имеется возможность позвонить человеку, в приложении реализован чат в реальном времени.

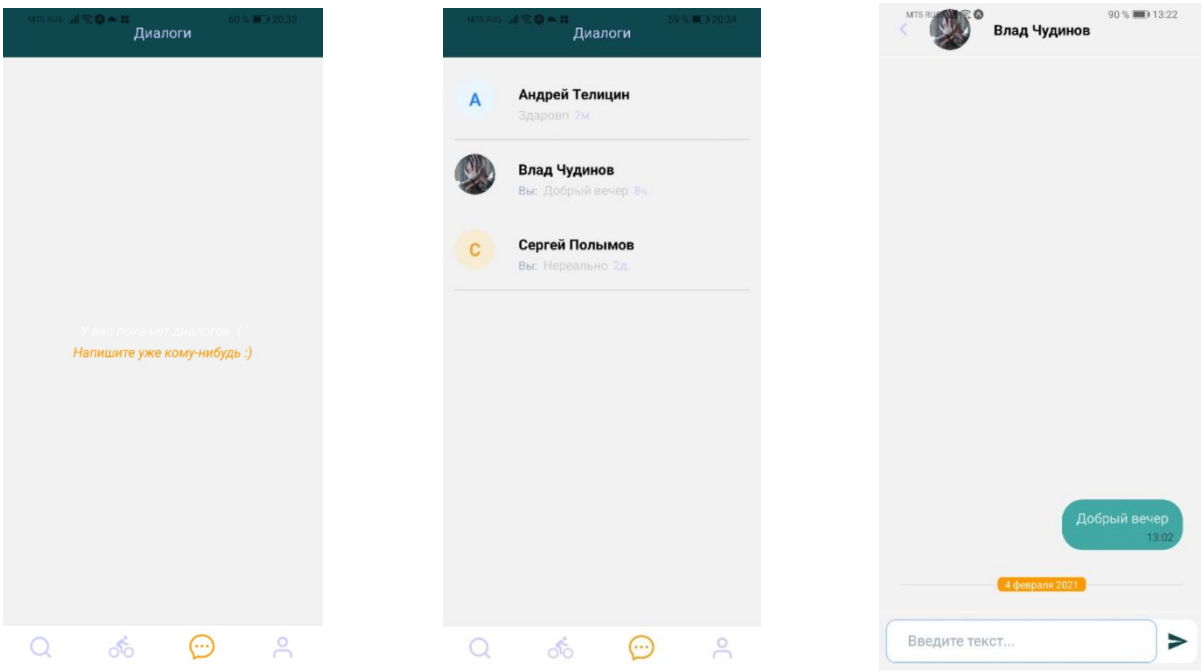

Рисунок 35 – диалоги и чат с пользователем

На первом изображении можно наблюдать вид экрана, при отсутствии диалогов. На втором их присутствие. Для открытия чата, который изображен на третьем изображении, необходимо нажать в любом месте конкретного диалога. На экране чата с пользователем, для удобства было спрятано навигационное меню, а для возврата на предыдущий экран, необходимо воспользоваться кнопкой в заголовке.

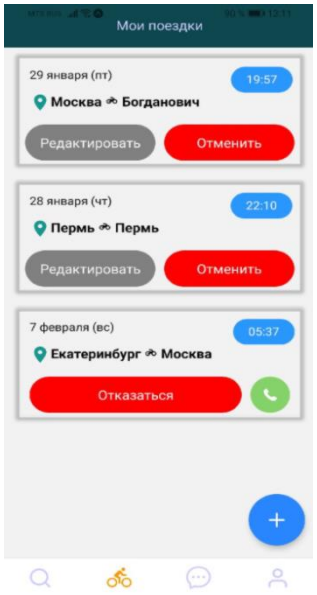

Рисунок 36 – экран мои поездки

На данном экране выводится в блочном виде информация о поездках, где текущий пользователь организатор или участник. И дополнительная кнопка по добавлению новой поездки. Отсюда появляется возможность отменить или отредактировать свои поездки, либо отказаться или позвонить организатору, где пользователь выступает в роли участника.

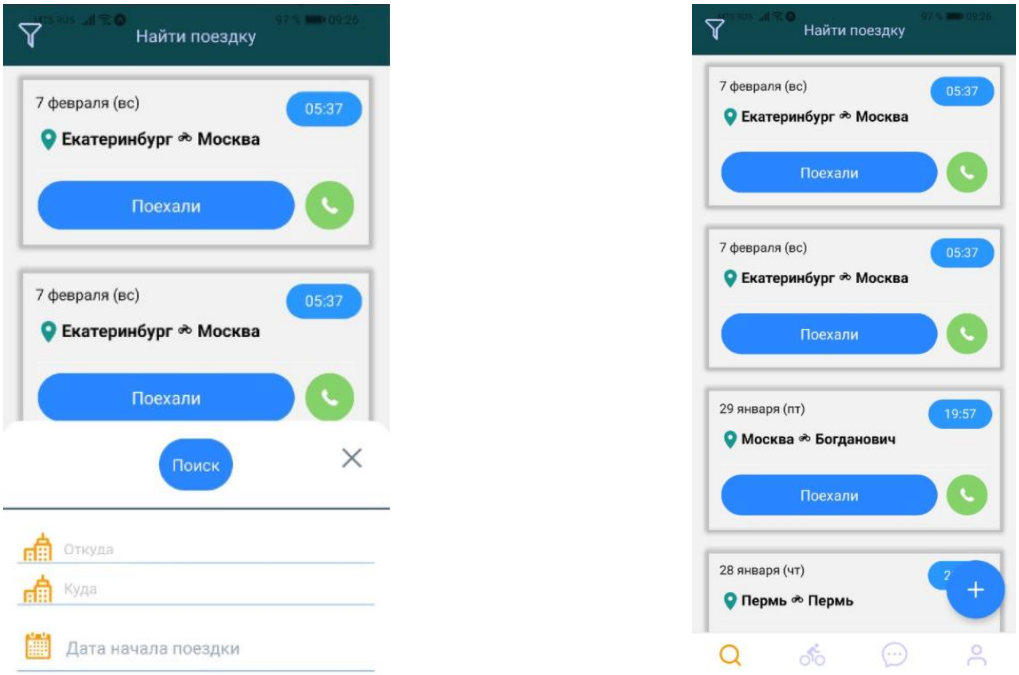

Рисунок 37 – экран всех маршрутов

Подобный экран с моим поездками, но назначение другое, выводит все поездки, дополнительно появилась возможность вызвать фильтр по поиск, путем нажатия соответствующей кнопки. По умолчания изначально выводятся маршруты, у которых отправная точка, совпадает с населенным пунктом, указанном в профиле пользователя. Поля «Откуда» и «Куда» заполняются путем выбора выпадающих значений из списка. У поля «Дата начала поездки» форма выбора даты из соответствующего календаря.

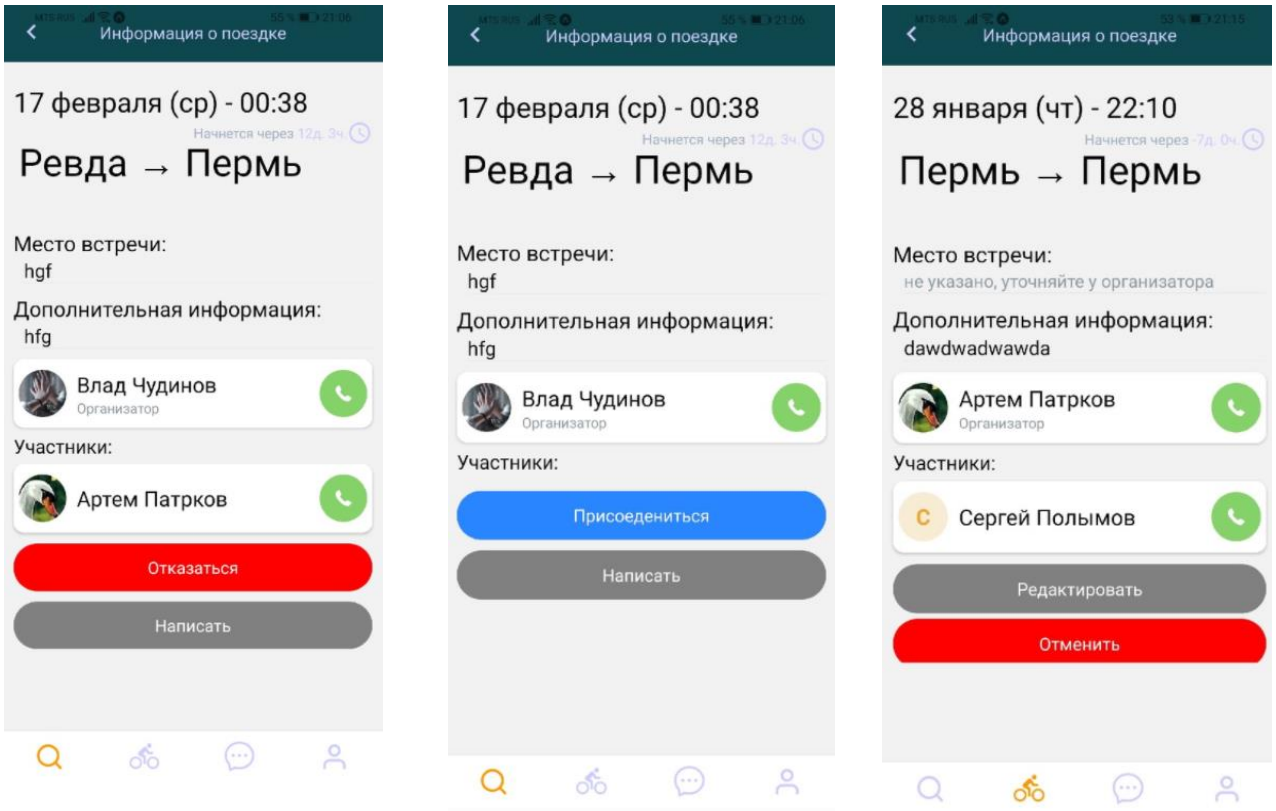

Рисунок 38 – информация о маршруте

При просмотре детальной информации, разница состоит не только в отображении информации, но в возможных выполняемых действиях, возможно присоединиться, написать организатору, редактировать или отменить, если текущий пользователь организатор, либо отказаться, если участник.

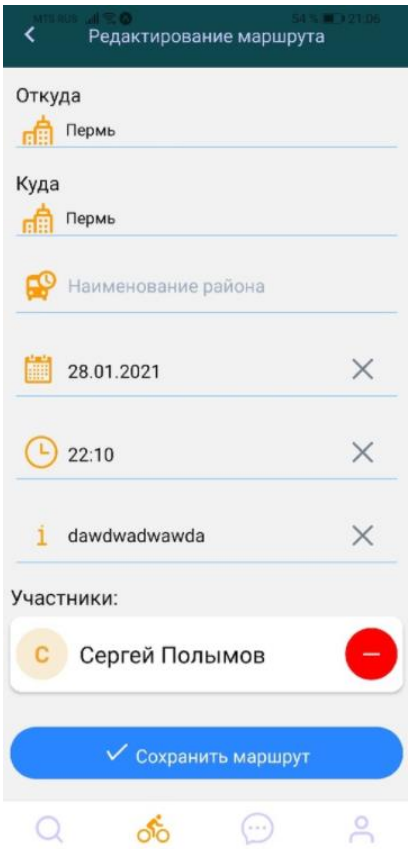

Рисунок 39 – редактирование маршрута

В режиме редактирования, возможно, не только изменить информацию, но также удалить уже присоединившихся пользователей, далее для таких пользователей она будет не видна. Сохранение информации вызывается нажатием соответствующей кнопки «Сохранить маршрут». Отменить редактирование возможно возвратом на предыдущий экран, при нажатии кнопки в заголовке, либо при использовании навигационной панели.

2.3 Техническая документация

2.3.1 Описание проекта интерфейса

Структура проекта:

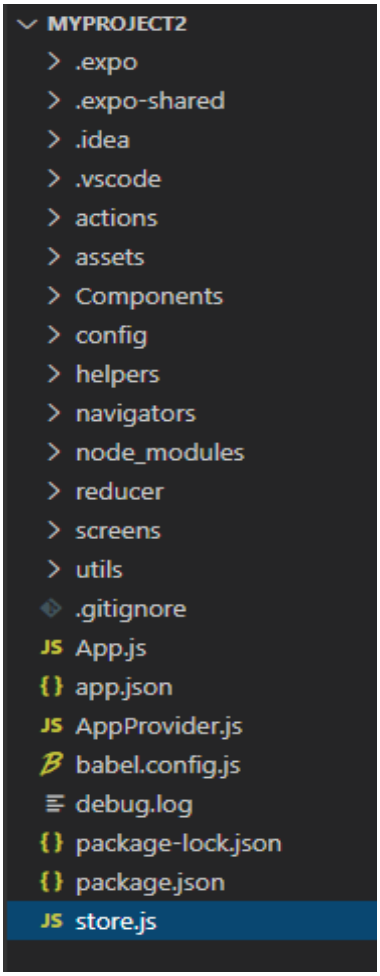

Рисунок 40 – структура проекта интерфейса мобильного приложения

Описание назначения основных папок проекта:

1. Actions – действия для запроса данных из базы данных.
2. Assets – хранение стилей компонентов.
3. Components – описание компонентов, используемых на экранах.
4. Config – настроечные файлы, конфигурация проекта.
5. Helpers – дополнительные функции для проекта.
6. Navigators – хранение настроек навигации экранов приложения.
7. Reducer – функции для хранения информации в состояния.

8. Screens – экраны приложения.
9. Utils – одноразовые функции, вынесенные из папки Helpers.

Описание экранов мобильного приложения:

1. AddTripScreen.js – экран с добавлением поездки.
2. AuthLoginScreen.js – экран авторизации.
3. AuthRegisterScreen.js – экран регистрации.
4. DialoguesScreen.js – экран диалогов.
5. EditMyProfileScreen.js – экран редактирования профиля.
6. EditTripScreen.js – экран редактирования поездки.
7. HomeScreen.js – экран со всем списком поездок.
8. InitialScreen.js – загрузочный экран, при запуске приложения, определяет нужно ли пользователю авторизация.
9. MessagesScreen.js – экран диалога с конкретным пользователем.
10. MyHomeScreen.js – экран с моими поездками.
11. MyProfileScreen.js – экран профиля текущего пользователя.
12. TripScreeen.js–экран информации о поездке.

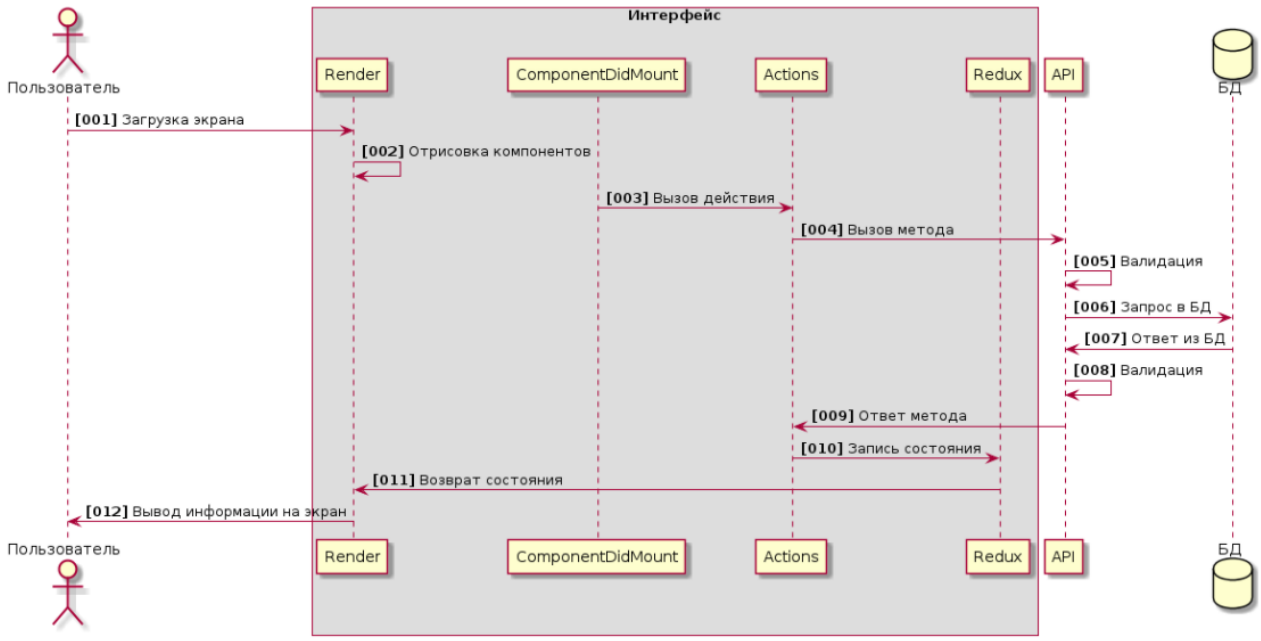

Рисунок 41 – диаграмма последовательности вывода информации на экран приложения

2.3.2 Описание проекта API

Структура проекта:

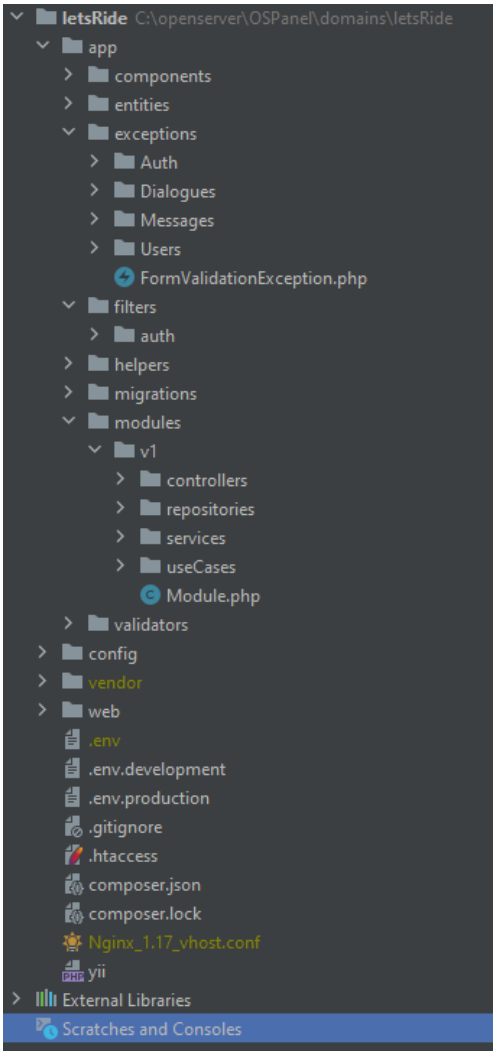

Рисунок 42 – структура проекта API

Описание назначения основных папок:

1. App:

1.1. Components – содержит настроечные данные, например информацию статуса ответа на запросы, класс оборачивания в транзакцию запрос к базе данных при необходимости и т.п.

1.2. Entities – содержит файлы с сущностями.

1.3. Exceptions – файлы с классами обработки исключений методов, в случае непредвиденной ситуации, для вывода понятной информацию на экран.

1.4. Filters – хранить в себе единственный файл с классом по работе с токеном пользователя, его проверку, обработку и т.д.

1.5. Helpers – содержит вспомогательные классы, так как формализация номера телефона, генерация уникального года диалога с пользователями.

1.6. Modules – содержит множество файлов, связанных с основной работы методов и базой данной. Хранить в себе преобразованные на PHP SQL запросы в базу данных для выборки данных. А также дополнительную проверку, с более глубокой сложностью обработки данных, полученных из базы данных.

1.7. Validators – класс для дополнительной валидации номера телефона.

1.8. Migrations – механизм отслеживания изменений в структуре базе данных.

2. Config – хранение конфигурационных файлов проекта.
3. Vendor – хранение зависимостей Фреймворка Yii
4. Web –хранение корневого файла «index.php» проекта.

Реализованы следующие методы:

1. /auth/login (GET) – авторизация.
2. /auth/logout (POST) – выход из учетной записи.
3. /auth/register (PSOT) – регистрация, добавление пользователя.
4. /account (GET) – получение информации о пользователе.
5. /account (POST) – изменения данных профиля.
6. /cities (GET) – получение списка населенных пунктов.
7. /chat/dialogues (GET) – получение диалогов.

8. /chat/users/id/messages (GET) – получение списка сообщений диалога.
9. /chat/users/id/messages (POST) – создание сообщения.
10. /users/id (GET) – получение информации о пользователе.
11. /trips (GET) – получение списка маршрутов.
12. /trips/id (GET) – получение информации по определенному маршруту.
13. /trips/id/remove (POST) – удаление маршрута.
14. /trips (POST) – создание маршрута.
15. /trips/id (POST) – изменение маршрута.
16. /account/bookings/create – присоединение к маршруту.
17. /account/bookings/remove – отмена в участие.
18. /account/bookings/reject – отклонение участника организатором.
19. /account/routes-bookings (Get) – получение участников.

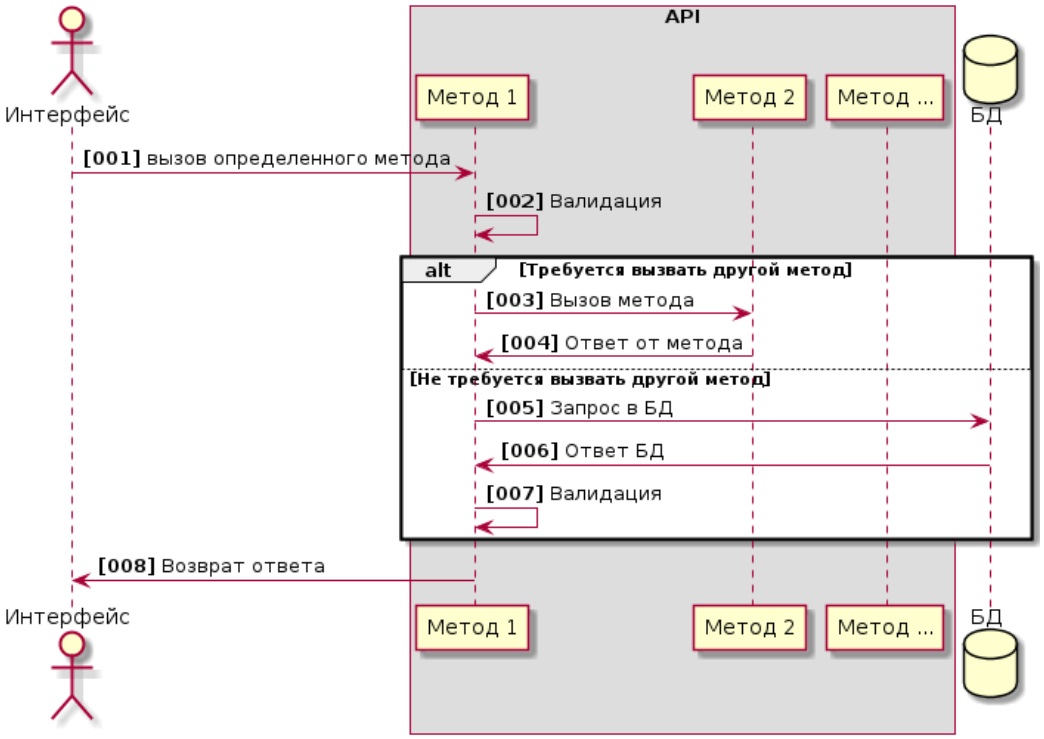

Рисунок 43 – диаграмма последовательности работы методов.

Заключение

В процессе разработки мобильного приложения и выполнения данной работы было выполнено:

1. Проведен анализ состояния проблемы и выявлены подходы ее решения. Была подтверждена актуальность разрабатываемого мобильного приложения.
2. Произведен анализ и обоснован выбор технологий реализации и программных платформ. Нами выбрано: язык программирования PHP, Фреймворк React Native, Фреймворк Yii, mySql, openServer, Ide Visual Studio Code, IdePhpStorm, хранилище исходников проекта и версионный контроль Git, операционная система Windows 10, Postman.
3. Разработано мобильное приложение на основании предоставленного технического задания.

Данное приложение позволяет выступать в роли организатора совместных велопоходов или быть участником, уже созданного. Также присутствует возможность оперативной коммуникации с организатором маршрута, посредством онлайн чата. Может быть использовано любым человеком, при наличии смартфона и установленной на нем, интерфейсной части мобильного приложения.

Список информационных источников

1. Семенчук В. Мобильное приложение как инструмент бизнеса – «Альпина Диджитал», 2016. – 270 с.
2. Молчанов А.Ю. Системное программное обеспечение: Учебник для вузов – СПб.: Питер, 2006. – 396 с.
3. Анохин Антон Борисович Android для телефонов и планшетов: недостающее руководство для всех! – М.: Изд-во Триумф, 2012. – 224 с.
4. Круг Стив Не заставляй меня думать. Веб-юзабилити и здравый смысл. 3-е изд. – М.: Эксмо, 2017. – 256 с.
5. Мартин Роберт Чистая архитектура. Искусство разработки программного обеспечения. – СПБ.: Питер, 2018. – 352 с.
6. Ричардсон Крис Микросервисы. Паттерны разработки и рефакторинга. – СПБ.: Питер, 2020. – 544 с.
7. Машнин Т.С. Web-сервисы Java. – СПБ.: БХВ-Петербург, 2012. – 560 с.
8. Дашнер С. Изучаем Java EE. Современное программирования для больших предприятий. – СПБ.: Питер, 2018. – 384 с.
9. Мотев А.А. Уроки MySQL. Самоучитель. – СПб.: БХВ-Петербург, 2006. – 208 с.
10. Вонг Уоллес Основы программирования для «чайников», 3-е издание. Пер. с англ. – М.: Издательский дом «Вильямс», 2004. – 384 с.
11. Машнин Т.С. Разработка Android-приложений в деталях. – СПБ.: БХВ-Петербург, 2020. – 665 с.
12. Алеев А. Быстрый старт Flutter-разработчика. – Издательское решение, 2020. – 247 с.
13. Томас Марк Тиленс React в действии. – СПб.: Питер, 2019. – 368 с.
14. Skillbox Популярные языки программирования бэкенда: сайт. – URL: https://habr.com/ru/company/skillbox/blog/534684 (дата обращения: 04.02.2021)

15. Тратчтенберг А., Скляр Д. PHP Cookbook – 3-е изд. O'Reilly Media, 2014. – 736 с.
16. Рейтц К, Шлюссер Т. Автостопом по Python. – СПБ.: Питер, 2017. – 336 с.
17. Фримен А. Angular для профессионалов. – СПБ.: Питер, 2018. – 800 с.
18. Хорсдал К. Микросервисы на платформе .NET. – СПБ.: Питер, 2018. – 352 с.
19. Монахов В.В. Язак программирования Java и среда NetBeans. – 3-е изд., перераб. и доп. – СПБ.: БХВ-Петербург, 2011. – 704 с.
20. Mukund Chaudhary, Ankur Kumar PhpStorm Cookbook. Packt Publishing, 2014. – 254 с.
21. Макаров А.С. Yii. Сборник рецептов. – М.: ДМК Пресс, 2013. – 372 с.
22. Симпсон К. ES6 и не только. – СПб.: Питер, 2017. – 336 с.
23. Прохоренок Н.А. HTML, JavaScript, PHP и MySQL. Джентельменский набор Web – мастера. – 3-е изд., перераб. и доп. - СПБ.: БХВ-Петербург, 2010. – 912 с.
24. Колисниченко Д.Н. PHP и MySQL. Разработка веб-приложений. – 6-е изд., перераб. и доп. - СПБ.: БХВ-Петербург, 2017. – 640 с.
25. Дронов В.А. Laravel. Быстрая разработка современных динамических Web-сайтов на PHP, MySQL, HTML и CSS.- СПБ.: БХВ-Петербург, 2017. – 768 с.
26. Локхарт Д. Modern PHP New Features and Good Practices. O'Reilly Media, 2015. – 267 с.
27. Фиртман М. jQuery Mobile: разработка приложений для смартфонов и планшетов: Пер. с англ. – СПб.: БХВ-Петербург, 2013. – 256 с.
28. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж. Паттерны объектно-ориентированного проектирования – СПБ.: Питер, 2020. – 448 с.
29. Эделман Дж., Лоу С. С., Осуолт М. Автоматизация программируемых сетей – М.: ДМК Пресс, 2019. – 616 с.
30. Гринт, Зак, Ньюман, Крис MySQL. Карманный справочник. : Пер. с англ – М.: ООО «И.Д.Вильямс», 2006. – 224 с.

31. Кудрявцев А.В. Генератор рабочих программ дисциплин на основе использования средств системы управления базами данных mysql. //Информатизация образования: теория и практика: сб. материалов Междунар. науч.-практ. 49 конф. (Омск, 17–18 ноября 2017 г.) /под общ. ред. М. П. Лапчика. – Омск: Изд-во ОмГПУ, 2017. – 420 с.